Le Programmeur

Microsoft® Visual Basic® 6

Créez des applications efficaces en VB6

Greg Perry

PEARSON

LE PROGRAMMEUR

Microsoft®

Visual Basic® 6

Greg Perry

PEARSON Education

Publié par Pearson

47 bis, rue des Vinaigriers

75010 PARIS

Tél. : 01 72 74 90 00

Réalisation PAO : TYPAO

ISBN : 978-2-7440-4082-5

Sommaire

Introduction

Tout au long de ces vingt et un chapitres, vous allez apprendre à développer des applications Windows en Visual Basic. Nous nous efforcerons de rendre cet apprentissage aussi peu rébarbatif que possible. La simplicité du langage et le caractère visuel de l'environnement font de Visual Basic un outil des plus conviviaux. En fait, la création d'une application en Visual Basic consiste en grande partie dans la disposition des contrôles qui formeront l'interface. Votre application Windows apparaît au fur et à mesure que vous ajoutez les objets, et telle qu'elle le sera à l'utilisateur. Visual Basic est, à ce propos, l'un des tout premiers langages de programmation à avoir intégré un environnement réellement WYSIWYG (*what you see is what you get*, ou tel écrit tel écran).

Même si vous n'avez encore jamais écrit une seule ligne de code, ces vingt et un chapitres sauront vous initier, dans la plus grande simplicité, aux techniques de programmation professionnelles. Chaque chapitre traite son sujet spécifique de fond en comble en l'étayant d'exemples de code, dans un esprit de travaux pratiques. Exercices et questions-réponses vous permettront d'affermir et de mettre en œuvre les notions étudiées dans chaque chapitre. Enfin, les Projets bonus répartis à travers l'ouvrage vous invitent à mettre "la main à la pâte", pour créer des applications complètes et fonctionnelles, et le fonctionnement est ensuite analysé ligne par ligne. Ainsi, chaque acquis est renforcé d'une mise en pratique. Vous créerez même votre premier programme Visual Basic dès le Chapitre 1 !

Enseigner la programmation Visual Basic au nouveau venu est une tâche délicate, en raison notamment d'une évolution parallèle des outils et des compétences que cela requiert. Certains en viennent à Visual Basic après avoir tâté de langages de programmation plus avancés — mais aussi plus fastidieux —, tels que C++. D'autres ont seulement l'expérience que de QBasic. QBasic est un langage fourni sur la plupart des PC depuis de nombreuses années. L'interface textuelle de MS-DOS, lente et peu commode, a valu à QBasic son obsolescence. Pourtant, QBasic n'est jamais qu'un cousin un peu éloigné de Visual Basic, et en constitue une excellente passerelle. Enfin, il

y a les vrais débutants, ceux qui n'ont jamais programmé. Pour ceux-là, Visual Basic n'est pas la seule nouveauté : ce sont les principes de la programmation elle-même qu'il faudra aussi leur faire découvrir.

Visual Basic est bien plus qu'un simple langage de programmation. Ce langage pilote en arrière-plan tout ce qui se passe pendant l'exécution du programme. Mais, pour le programmeur comme pour l'utilisateur, c'est l'interface visuelle qui prime. Les applications Windows offent aux utilisateurs un haut degré d'interaction, grâce au système de fenêtres et aux divers objets graphiques. Le code peut être aussi efficace que l'on voudra, si l'interface n'est pas satisfaisante, l'utilisateur n'aimera pas votre programme. Vous serez harcelé de demandes d'aide et de support. Et vos clients pourraient hésiter à acheter les mises à jour ultérieures.

C'est pourquoi nous insisterons, particulièrement au début de l'ouvrage, sur cette question de l'interface utilisateur. Une fois que vous serez capable de concevoir et de créer une interface viable et attrayante, alors seulement vous commencerez à étudier la mécanique interne des programmes.

Notre monde change rapidement. Les entreprises se renouvellent, les secteurs se restructurent, les activités se transforment. Vos programmes doivent suivre le mouvement. Ils doivent être flexibles et aisés à maintenir, afin de pouvoir satisfaire à tous les changements demandés par le client. Nous insisterons donc également sur les étapes de conception, d'écriture, de test et de maintenance qui font un programme de qualité. Un programme est écrit une fois, mais actualisé de nombreuses fois. En suivant les principes que nous vous indiquerons quant à l'écriture du code, vous vous épargnerez pas mal de peine au moment de la maintenance et de la mise à jour.

Entre théorie et pratique, nous tenterons de vous faire découvrir le plus d'aspects possible de Visual Basic, tout en prenant soin d'éviter les détails qui ne concernent pas le programmeur Visual Basic "normal". Nous nous attacherons à l'essentiel : vous aider à construire des programmes efficaces, concis, clairs, documentés et faciles à maintenir.

Voici, entre autres, ce que vous apprendrez à :

- construire une interface utilisateur appropriée ;
- jeter les bases du programme avec l'assistant Création d'applications ;
- structurer le code de sorte que le programme s'exécute sans accrocs ;
- comprendre les outils les plus courants de l'environnement Visual Basic ;
- maîtriser l'art du débogage ;
- intégrer à vos applications les technologies de bases de données ;
- intégrer l'accès Internet au cœur de vos programmes ;

- gérer les contrôles ActiveX qui permettent à Visual Basic d'exploiter des outils venus d'autres langages ou applications ;
- créer vos propres objets d'interface grâce aux fonctionnalités ActiveX de Visual Basic ;
- accéder au moteur d'aide en ligne pour faciliter la tâche des utilisateurs ;
- offrir, grâce aux boîtes de dialogue communes, une interface familière à vos utilisateurs ;
- permettre à l'utilisateur d'exécuter une commande d'un simple clic, par le biais des barres d'outils et des coolbars ;
- programmer les API Windows qui permettront à Visual Basic d'exploiter des fonctionnalités Windows supplémentaires ;
- dynamiser vos applications à l'aide des images et du multimédia.

Partie I

D'un coup d'œil

Cette partie vous apprend à manœuvrer dans l'environnement Visual Basic, à créer les éléments visuels de votre application Windows, et vous enseigne les fondements du langage de programmation Visual Basic.

Dès le premier chapitre, vous allez créer une application semblable aux applications Windows que vous avez déjà eu l'occasion d'utiliser. Les chapitres suivants vous font découvrir les différents objets et techniques qui rendront vos programmes plus puissants. A la fin de chacun de ces chapitres, une série de questions et d'exercices vous permet de récapituler vous-même les points importants et de les mettre en pratique.

La programmation exige plus que la simple connaissance du langage et de ses commandes. En progressant dans les chapitres qui suivent, vous comprendrez mieux l'importance d'écrire des programmes clairs et bien documentés. Le contexte dans lequel les utilisateurs exploitent vos applications change, et vos applications doivent changer également. En suivant les principes et exemples étudiés dès le début, vous apprendrez à écrire des programmes faciles à maintenir.

Visual Basic sert à concevoir des programmes Windows. Dans cette partie, nous passerons en revue tous les éléments (ou presque) qu'implique cette interface. Vous apprendrez ainsi à disposer des contrôles sur la feuille, à créer et à répondre aux menus, ainsi qu'à gérer les autres types d'interactions entre l'utilisateur et le programme. Vous commencerez alors à maîtriser le cœur de Visual Basic : le langage de programmation.

Visual Basic est l'un des outils de programmation les plus conviviaux. Le développement d'une application consiste en grande partie à disposer des objets graphiques et à en régler le comportement à l'aide de propriétés. Visual Basic est en fait le seul vrai langage de programmation que les débutants puissent apprendre si facilement. Il permet également aux programmeurs avancés de créer des applications Windows puissantes.

Que vous soyez novice ou un peu expérimenté, vous serez surpris de découvrir ce que Visual Basic peut faire pour vous, et ce que vous pouvez faire avec Visual Basic.

Chapitre 1

Présentation de Visual Basic

Visual Basic 6 est la dernière — et la meilleure — version du langage de programmation Visual Basic de Microsoft. Si l'écriture de programmes est parfois une assommante corvée, Visual Basic réduit les efforts à fournir et peut faire de cette activité une partie de plaisir. Avec Visual Basic 6, la plupart des tâches de programmation deviennent aussi simples que de déplacer des objets graphiques à l'aide de la souris.

Nous commençons ici un enseignement de Visual Basic en vingt et un jours. Avant la fin de ce chapitre, vous aurez créé votre toute première application Visual Basic. Au terme des trois prochaines semaines, vous maîtriserez Visual Basic 6 et serez en mesure de développer des applications efficaces selon vos besoins.

Voici ce que nous étudierons aujourd'hui :

- l'histoire de Visual Basic ;
- les méthodes et les processus de programmation ;
- comment l'interface visuelle de Visual Basic rend la programmation simple et amusante ;
- l'assistant Création d'applications ;
- pourquoi la programmation événementielle est si importante en environnement Windows.

Les dessous de Visual Basic

Une fois saisis les fondements de Visual Basic, vous serez à même d'en exploiter tous les ressorts. Microsoft a construit Visual Basic sur la base d'un *langage de programmation* pour débutants : le BASIC. Sous une forme ou une autre, le BASIC est utilisé depuis plus de trente-cinq ans. L'objectif de ses premiers concepteurs était de développer un langage de programmation accessible aux novices. Avec le BASIC, les programmeurs en herbe devenaient rapidement efficaces. Les autres langages de programmation en usage à l'époque, tels que le COBOL, le FORTRAN ou l'Assembleur, nécessitaient un apprentissage beaucoup plus poussé avant d'être réellement productif.

BASIC est l'acronyme de Beginner's All-purpose Symbolic Instruction Code (code d'instructions symboliques multifonction pour débutants). Ça parle de soi-même !

Un langage de programmation est un ensemble de commandes et d'options de commandes (les arguments) par lequel on envoie des instructions à l'ordinateur. Les ordinateurs ne peuvent pas (pas encore) comprendre le langage des humains, d'abord parce que les humains peuvent réagir à des instructions ambiguës, ce qui est foncièrement impossible pour la machine. Un langage de programmation doit être plus précis qu'un langage naturel.

Les langages de programmation sont plus faciles à apprendre que les langues étrangères. Les langages pour ordinateurs comprennent souvent moins de 300 commandes, commandes qui renvoient à des syntaxes ou des concepts familiers même aux non-anglophones : `Open` *(ouvrir),* `Next` *(suivant), etc.*

Bien que ce langage fût conçu pour les débutants, les programmes BASIC restaient quelque peu ésotériques, et demandaient malgré tout un apprentissage. Le Listing 1.1 montre un exemple de programme écrit en BASIC, dont le but est d'afficher le carré des nombres de 1 à 10. Même si vous pouvez, avec quelques rudiments de langue anglaise, en percer les grandes lignes, ce programme n'est certainement pas ce qui se fait de plus clair ; une compréhension minimale du BASIC est requise pour pleinement comprendre la raison d'être et l'articulation de ses éléments.

Les programmes sont souvent constitués de plusieurs programmes interagissant les uns sur les autres ; c'est pourquoi on désigne souvent par application l'ensemble des fichiers d'un programme. Le programme, ou application écrite dans un langage de programmation, est un jeu d'instructions qui dirige l'ordinateur.

Listing 1.1 : Les premiers programmes BASIC, où l'on numérotait les lignes, étaient quelque peu ésotériques

```
10 REM Ce programme calcule et affiche les 10 premiers carrés.
20 CLS
30 PRINT "Carrés de 1 à 10"
40 PRINT "Valeur", "Carré"
50 FOR N = 1 TO 10
60   PRINT N, (N*N)
70 NEXT N
80 PRINT
90 END
```

Ne pas faire
Ne paniquez pas pour des histoires de carrés. Vous n'aimez pas les maths ? Pas de problème : Visual Basic fera à votre place tous vos devoirs de calcul.

Si vous lanciez le programme BASIC, vous obtiendriez cette sortie :

```
Carrés de 1 à 10
Valeur       carré
1            1
2            4
3            9
4            16
5            25
6            36
7            49
8            64
9            81
10           100
```

Notez que le BASIC est un langage strictement textuel. L'entrée et la sortie des données se font en mode texte ; les fenêtres et autres graphismes sont l'apanage des programmes modernes.

Visual Basic n'a pas été créé directement depuis le BASIC original. Bien que, en trente-cinq ans, le BASIC ait évolué sous beaucoup de rapports, l'essentiel de sa structure a été conservé. Le BASIC fut choisi par Microsoft comme langage de programmation principal du premier système d'exploitation MS-DOS ; mais il a fallu le perfectionner et l'enrichir de nouvelles fonctionnalités, jusqu'à le rééditer sous de multiples avatars : MBASIC (pour Microsoft BASIC), GWBASIC, BASICA (pour BASIC avancé), QuickBASIC, et enfin Qbasic — encore fourni sur les CD-ROM Windows.

A travers cette évolution, le langage BASIC a gardé sa nature simple, tout en s'enrichissant continuellement de nouvelles et puissantes commandes. Le caractère "texte pur" de langages comme QBasic permet aux programmeurs novices d'acquérir une certaine vitesse de travail plus facilement qu'avec la plupart des langages graphiques, comme Visual C++. Afin de maintenir cette facilité d'utilisation, Microsoft a voulu garder aux différentes versions du BASIC leur nature *interprétée*, opposée aux langages *compilés*. Avec un langage interprété, le programmeur peut exécuter immédiatement son programme, voir sans délais les résultats et les éventuelles erreurs. Ce *feedback* instantané est capital pour les débutants, qui ont besoin de réponses rapides quand ils apprennent à programmer. Les langages compilés s'exécutent plus rapidement et sont mieux appropriés au développement d'applications commerciales, mais requièrent beaucoup plus d'efforts et d'apprentissage.

Les langages interprétés, tel que le BASIC, vous permettent d'exécuter le programme à tout moment, alors que vous l'écrivez. Cette rapidité de la "réponse" en font de bons points de départ pour l'apprentissage. Les langages compilés nécessitent des étapes supplémentaires, la compilation et la liaison, avant que le programmeur ne puisse exécuter son œuvre. La compilation transpose le programme du langage dans lequel il a été écrit au langage natif de l'ordinateur.

Alors que Windows devenait plus populaire, Microsoft a réalisé que QBasic, langage purement textuel, n'était pas exploitable comme langage de programmation "fenêtrée". On a donc développé Visual Basic, langage fondé sur le BASIC, mais bien mieux adapté aux environnements modernes. A la différence de QBasic et des autres avatars du BASIC, tous textuels, Visual Basic est un langage visuel. Bien qu'on rencontre aussi en Visual Basic du *code* plus ou moins semblable au programme du Listing 1.1, la plus grande partie d'un programme Visual Basic est constituée d'éléments graphiques sans aucune ressemblance avec le code "à l'ancienne". La Figure 1.1 montre un écran incluant de nombreux éléments de programmation Visual Basic.

Le code est un autre mot servant à désigner l'ensemble d'instructions du programme.

Bien avant d'avoir refermé ce livre, vous serez capable de déchiffrer tous les éléments de la Figure 1.1. Le fouillis apparent de l'écran déconcerte un peu au début, mais la simplicité de Visual Basic prend vite le pas.

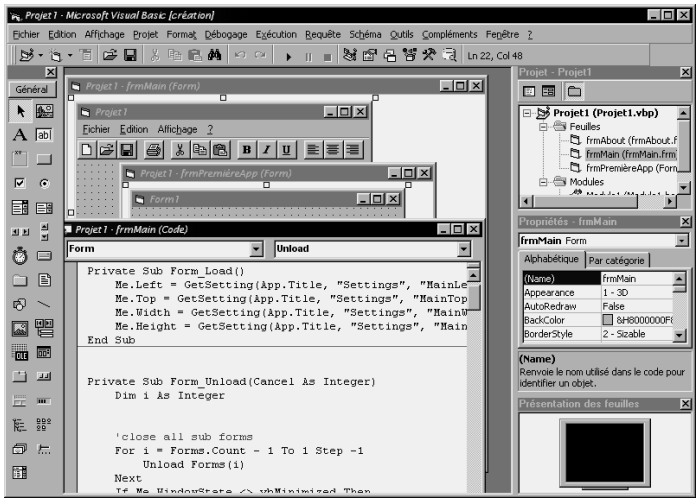


Figure 1.1
L'écran de programmation Visual Basic peut paraître un peu encombré, mais il s'avère très simple d'utilisation.

Si Visual Basic est devenu l'un des langages les plus utilisés, c'est que, en plus d'être graphique et simple d'utilisation, il est à la fois interprété *et* compilé. A mesure que vous écrivez votre programme Visual Basic, vous pouvez le tester en l'exécutant de façon interprétée, jusqu'à ce que tous les bogues aient été isolés et éliminés. Une fois le programme dûment testé et tous les problèmes réglés, il ne reste qu'à compiler un exécutable rapide et sûr (personne ne peut modifier facilement le programme), prêt à être distribué aux utilisateurs. En faisant de la compilation une simple option de menu, Visual Basic prend sur lui les étapes les plus délicates de la compilation (notamment un procédé cabalistique du nom d'*édition de liens*), que les autres langages obligent à effectuer soi-même.

Un bogue (de l'anglais bug) est une erreur dans le programme. Si votre programme ne s'exécute pas correctement, il vous faudra le déboguer, c'est-à-dire éliminer une à une toutes les erreurs.

A l'époque où Microsoft a sorti la première version de Visual Basic, beaucoup prédisaient la déchéance du langage BASIC (ainsi que de ses rejetons, comme QBasic). Ces mauvaises langues considéraient qu'un langage fondé sur le BASIC ne pouvait servir à la programmation sérieuse, justement parce que ces personnes médisantes n'avaient jamais regardé le BASIC comme un langage sérieux. Les langages tels que C, C++ et Pascal faisaient alors fureur, en raison de leurs possibilités de compilation et aussi parce que leurs structures de programmation s'adaptaient plus facilement à l'environnement

Windows. Avec Visual Basic, Microsoft a donné à la communauté des programmeurs les leçons suivantes :

- Un langage de type BASIC peut être à la fois simple à comprendre et puissant.
- Avec une interface adéquate, un langage de type BASIC peut très bien fonctionner en environnement Windows.
- Visual Basic peut être tantôt langage interprété, tantôt langage compilé, selon les besoins du programmeur.
- Loin d'être obsolète, un langage fondé sur le BASIC peut devenir le langage le plus utilisé dans le monde.

La nature visuelle de Visual Basic

Comme vous avez pu le voir sur la Figure 1.1, Visual Basic 6 est plus qu'un simple langage de programmation. Le secret de Visual Basic tient dans son nom : *visual*. Dans les systèmes d'exploitation Windows d'aujourd'hui, les programmes doivent être capables d'interagir *graphiquement* avec l'écran, le clavier, la souris et l'imprimante. Les langages de programmation plus anciens, comme le BASIC, étaient parfaits dans des environnements purement textuels, mais sont tout à fait inappropriés aux interfaces graphiques des ordinateurs modernes.

Durant cette première partie, vous ne verrez pas grand-chose du langage de programmation Visual Basic comme tel, parce que la procédure de programmation en Visual Basic engage beaucoup plus d'interactions avec l'environnement visuel que de tâches d'écriture de code. C'est seulement lorsque vous aurez à créer des programmes plus avancés qu'il vous faudra apprendre, du langage, plus que les quelques commandes traitées lors des premiers chapitres.

Ce n'est pas seulement le langage BASIC sous-jacent qui rend Visual Basic facile à apprendre et à utiliser. La plus grande part du développement d'un programme Visual Basic revient à glisser-déposer (avec la souris) des éléments visuels sur l'écran. Au lieu d'écrire des séries complexes d'instructions d'entrée et de sortie pour gérer les interactions avec l'utilisateur, il vous suffira de faire glisser à travers l'écran des contrôles, tels que zones de texte et boutons de commande ; Visual Basic se charge de faire fonctionner proprement les contrôles lorsque l'utilisateur exécutera le programme.

L'utilisateur est celui qui utilise le programme. Vous, le programmeur qui écrit les programmes, êtes aussi utilisateur de vos propres programmes et de ceux qu'écrivent les autres. Le sytème de programmation Visual Basic n'est rien d'autre qu'un programme dont on se sert pour créer d'autres programmes.

Visual Basic est disponible en plusieurs éditions :

- **Visual Basic Edition Entreprise.** Créée pour la programmation en équipe et les environnements client-serveur, où le traitement et les données sont distribués à plusieurs ordinateurs.
- **Visual Basic Edition Professionelle.** Conçue pour les programmeurs professionnels qui souhaitent exploiter pleinement l'environnement de programmation Visual Basic. Cette édition inclut un jeu complet d'outils et d'assistants pour l'empaquetage et la distribution des applications. Nous supposons, dans cet ouvrage, que vous utilisez l'Edition professionnelle, comme la plupart des programmeurs Visual Basic. Si toutefois vous utilisez une autre version, la plupart des informations du livre s'appliquent également ; en effet, loin de nous être centrés exclusivement sur les outils de l'Edition professionnelle, c'est un tour d'horizon complet du langage et de l'environnement de programmation Visual Basic que nous vous proposons.
- **Visual Basic Edition Initiation.** L'essentiel de Visual Basic, avec un jeu complémentaire d'outils standards — tout ce dont vous avez besoin pour vous lancer dans la programmation Visual Basic. Vous trouverez, dans la documentation complète du Microsoft Developer Network (réseau des développeurs Microsoft), toute l'aide nécessaire à l'apprentissage et à l'utilisation de Visual Basic.

La version spéciale INFA de Visual Basic est livrée avec le coffret Visual Studio. Visual Studio est un environnement de programmation qui supporte plusieurs langages Microsoft, dont Visual Basic, Visual C++ et Visual J++. L'environnement Visual Basic est identique à celui qu'emploient les utilisateurs d'autres langages. Si vous vous essayez à un autre langage de programmation Microsoft, vous n'aurez donc pas à maîtriser un nouveau système de menus et de boîtes de dialogue.

Pourquoi écrire des programmes ?

La plupart des utilisateurs d'ordinateur n'auront jamais à apprendre un langage de programmation. Ils se contentent, en général, d'acheter leurs logiciels dans les magasins spécialisés, et ne ressentent jamais le besoin de programmes plus spécialisés. En revanche, il est difficile de trouver sur le marché un programme qui convienne exactement à une tâche spécifique, notamment lorsqu'on utilise l'ordinateur à des fins professionnelles ou scientifiques. Vous pouvez imaginer un nouveau concept de jeu qui, une fois concrétisé en logiciel best-seller, vous permettrait de partir en préretraite aux Baléares. Si l'on a besoin d'une application spécifique qui n'est nulle part disponible sur le marché, ou si l'on souhaite créer des logiciels pour gagner de l'argent, il faut concevoir et écrire ces programmes à l'aide d'un langage de programmation comme Visual Basic.

Souvenez-vous : vous ne pouvez simplement dire à l'ordinateur ce qu'il doit faire et attendre qu'il fasse le boulot pour vous. L'ordinateur n'est qu'une machine stupide et sans initiative ; il a besoin pour travailler que vous lui fournissiez une liste détaillée d'instructions. Vous lui communiquez ces instructions sous la forme d'un programme. Un programme Visual Basic est constitué de codes (semblables à celui du Listing 1.1) et d'éléments visuels qui définissent l'aspect de l'écran et les contrôles Windows avec lesquels l'utilisateur interagit lorsqu'il lance le programme.

En apprenant Visual Basic, vous apprenez aussi à automatiser les applications courantes comme celles que l'on trouve dans Microsoft Office. Microsoft Office est composé de plusieurs programmes qui travaillent enemble : traitement de texte, tableur, base de données, etc. Microsoft Office contient également une version complète du langage de programmation Visual Basic, grâce auquel on peut automatiser les applications Office. (Microsoft Office 97 intègre Visual Basic pour Applications 5, version de Visual Basic destinée au développement de sous-programmes pour les logiciels de la suite Office.) Par exemple, vous pouvez automatiser votre comptabilité mensuelle en écrivant un programme qui consolide vos feuilles de calculs Excel. Le Visual Basic qui est livré avec les applications Office n'est pas le système de développement complet de Visual Basic 6, mais il inclut une version complète du langage qui vous permet de contrôler au plus près les applications.

Le processus de programmation

Avec le temps, vous établirez votre propre façon d'écrire les programmes selon vos besoins. Cependant, il convient de suivre ces quelques règles et étapes lorsque vous programmez en Visual Basic :

1. Déterminer ce que votre application devra faire en créant un schéma général.
2. Créer la partie visuelle de votre application (les écrans et les menus avec lesquels l'utilisateur interagira).
3. Ajouter le code en langage Visual Basic qui reliera tous ces éléments visuels et qui automatisera le programme.
4. Tester l'application afin de déceler et d'éliminer tous les bogues.
5. Une fois les tests effectués, compiler le programme et distribuer l'application compilée aux utilisateurs.

Même si vous avez testé votre programme pour le purger de tous ses bogues avant de distribuer l'application, il est toujours possible qu'un utilisateur découvre un ou plusieurs nouveaux bogues. Les tests les plus approfondis ne garantissent jamais l'absence définitive de bogues. Plus votre programme est complexe, plus il fait de choses, plus grandes sont les chances qu'un bogue pointe sa vilaine tête au moment où vous et vos utilisateurs vous y attendez le moins. Malgré cette foncière impossibilité de retirer tous les bogues, il faut tester, tester… et tester encore. Eprouver toutes les possibilités du programme pour touver le plus grand nombre possible de bogues avant de compiler et de distribuer l'application.

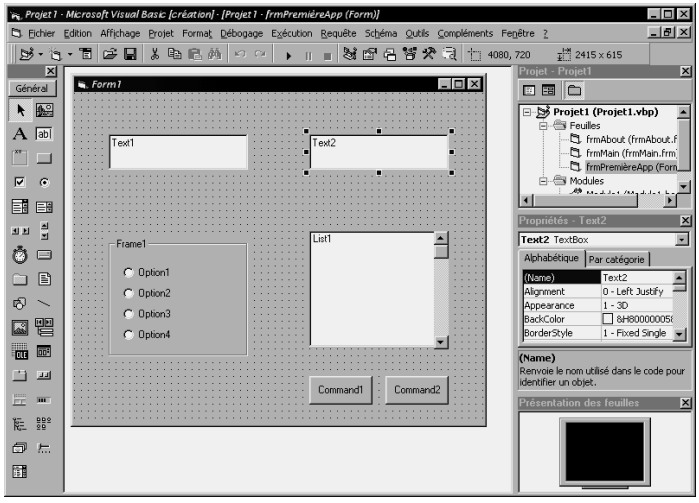
En attendant que votre application Visual Basic soit minutieusement et totalement testée, avant compilation, vous pouvez aider à accélerer le processus. En testant le programme de façon interactive, vous pouvez localiser et corriger les bogues plus facilement et plus rapidement. Visual Basic inclut un système d'aide spécial, appelé *débogueur*, qui vous aide à circonscrire les bogues révélés lors des tests. Vous apprendrez à vous servir du débogueur au Chapitre 21.

Un débogueur est un système de monitorage interactif, qui peut être activé ou désactivé à l'intérieur de Visual Basic, et qui vous aide à isoler les instructions fautives. Par exemple, si vous exécutez le programme que vous avez écrit et qu'il renvoie un résultat incorrect, le débogueur vous permet d'isoler rapidement la section du programme qui contient le bogue.

Avant Visual Basic, l'écriture de programme constituait une tâche fastidieuse sous plusieurs rapports. Dans un environnement textuel, il fallait d'abord dessiner sur le papier tous les écrans qui seraient affichés. Il fallait ensuite montrer tout cela aux utilisateurs pour vérifier que les dessins correspondaient exactement à leurs souhaits. Si vous conceviez un programme destiné à une distribution de masse, comme un jeu ou une application commerciale "généraliste", vous deviez coucher sur le papier tous les écrans, créer de complexes flux de données vers, et depuis, ces divers écrans, créer les fichiers résidents nécessaires à l'exécution du programme, et planifier à peu près chaque détail avant même de vous mettre au clavier.

Grâce à la nature visuelle de Visual Basic, vous ne touchez au clavier que bien plus tard dans le processus de programmation. Au lieu d'utiliser du papier, vous dessinez directement vos écrans à l'aide des outils Visual Basic. La Figure 1.2 présente un écran de ce genre. Pour créer un tel écran, aucun code n'est nécessaire : tout ce que vous avez à faire, c'est de faire glisser les divers contrôles sur la *fenêtre Feuilles*.

Figure 1.2
Visual Basic vous permet de concevoir et de créer les écrans à mesure que vous développez le programme.



La fenêtre Feuilles, appelée aussi "feuille", présente l'arrière-plan d'un écran de programme Visual Basic et inclut des éléments tels que boutons de commande et barres de défilement. Selon la nature et la complexité du programme, plusieurs fenêtres de feuilles peuvent être requises.

Vous pouvez tester vos écrans de programme (chaque feuille représentant la trame d'un écran) avant même d'ajouter le code ; en effet, Visual Basic vous permet d'exécuter interactivement le programme dès la création de la première feuille. Vous pouvez ainsi

vous assurer de l'élégance de l'écran, et montrer un *prototype* aux commanditaires du programme, qui ne se gêneront pas pour vous en donner un avis. A ce stade de prototype précodage, il est beaucoup plus simple de modifier le programme lorsque que le code a été ajouté. Cette fonction de prototypage est l'un des caractères de Visual Basic qui vous permettent de programmer rapidement et efficacement.

Un prototype est un programme d'essai qui ne contient presque aucune fonctionnalité, mais qui présente tout ou partie des écrans qui composeront le produit fini. Ce prototype est testé par vous et par les utilisateurs finals du programme afin de déterminer si les écrans contiennent bien tous les éléments requis.

Vous pouvez toujours apporter de nouvelles modifications au programme, même après que vous l'avez créé, testé, compilé et distribué aux utilisateurs. Mais ce serait une procédure fastidieuse, et qui impliquerait de distribuer de nouveau aux utilisateurs tous les fichiers de l'application. Dans tous les cas, plus tôt vous isolez les problèmes, plus il est simple d'y remédier.

La maintenance du programme

Les bogues ne constituent pas la seule raison qui puisse vous pousser à modifier encore un programme, alors que vous croyez en avoir terminé. La *maintenance du programme* est indispensable du fait que les exigences changent, les entreprises changent, les lois changent. Vous devez également retoucher le programme afin qu'il reste viable et d'actualité ; vous devrez régulièrement le mettre à jour pour prendre en compte les changements qui l'affectent. En outre, les utilisateurs auront toujours de nouvelles exigences sur ce que le programme doit faire.

La maintenance du programme est le terme employé pour désigner la mise à jour du programme après sa distribution. Cette mise à jour peut être le fait d'une demande des utilisateurs ou d'un changement dans la façon dont le programme doit opérer.

En fait, un programme est écrit une fois, mais retouché de nombreuses fois. Mieux vous assurez cette maintenance du programme, plus votre programme sera actuel et efficace. Vous pouvez décider de distribuer une nouvelle version du programme, avec un numéro de version différent qui s'affiche sur l'écran de démarrage et informe les utilisateurs de la récence du logiciel installé sur leur système.

Il convient de "documenter" votre code afin que d'autres programmeurs puissent comprendre votre code s'ils devaient le modifier par la suite.

A mesure que vous avancerez dans votre découverte du langage de programmation Visual Basic, vous apprendrez à écrire clairement votre code, ainsi qu'à établir la *documentation* du programme. Plus vous ajoutez de *commentaires* à votre programme, plus vous écrivez clairement le code au lieu d'user d'instructions obscures et alambiquées, plus il vous sera facile, ainsi qu'à d'autres, de traquer des erreurs et d'entretenir le programme.

La documentation est en fait une description du programme. Vous pouvez insérer la documentation dans le programme lui-même, de sorte que quiconque sera appelé à le modifier pourra comprendre chaque section du programme sans avoir à en deviner le sens. Les descriptions internes au programme Visual Basic sont appelées commentaires.

Il convient d'ajouter les *commentaires* à mesure que vous écrivez le programme, car c'est là que vous êtes le plus à même de le comprendre et de le décrire. Si vous attendez d'avoir terminé l'application, comme le font beaucoup de programmeurs, cette application risque de ne jamais être correctement documentée ; en effet, d'autres projets s'imposeront à vous, et les tâches documentation sont souvent mises de côté une fois qu'un projet est achevé.

A titre complémentaire, il peut être utile d'établir une documentation externe, augmentée de captures des différents écrans du programme et d'indications sur la façon dont l'utilisateur lance, utilise et quitte le programme. Plus complète sera la documentation que vous leur fournirez, plus facilement les utilisateurs maîtriseront votre application ; et plus ils feront de bons clients.

Votre premier programme

Si vous êtes familier des autres produits Windows, comme Microsoft Publisher, vous avez sans doute déjà vu à l'œuvre les *assistants*, qui vous aident et vous guident dans la création de documents selon vos besoins. Visual Basic dispose aussi d'assistants qui vous aident à créer votre programme. Pour écrire un programme Visual Basic, vous avez le choix de partir de zéro ou de créer la charpente de l'application à l'aide d'un assistant. Une fois que l'assistant a établi la structure générale du programme, il ne vous reste qu'à le compléter de tous les détails.

L'assistant pose une série de questions et sollicite de vous des précisions. A mesure que vous répondez, l'assistant génère l'application selon les critères que vous avez spécifiés. Visual Basic propose plusieurs assistants, mais celui que vous invoquerez sans doute le plus souvent est l'assistant Création d'applications.

Il est parfois difficile de choisir entre créer l'application ex nihila et établir la structure du programme à l'aide d'un assistant pour ensuite le finaliser selon les exigences du projet. Si l'on a déjà développé une application semblable à celle qui est requise, on peut simplement faire une copie de la première et la retravailler selon les nouveaux impératifs. Avec le temps, vous apprendrez à apprécier les situations et à opérer les bons choix.

Pour vous aider dans vos premiers pas, cette section vous guidera dans la création de votre toute première application. Vous découvrirez avec quelle facilité l'assistant Création d'applications vous permet de définir la structure du programme. Bien que l'application résultante ne soit pas vraiment exploitable comme telle (après tout, ce n'est qu'un squelette), vous serez surpris de tout ce que l'assistant Création d'applications peut créer automatiquement. La leçon de demain vous enseignera à créer une application ex nihilo, sans recourir à l'assistant Création d'applications.

Cela peut paraître surprenant, mais vous créerez sans doute beaucoup plus souvent vos applications ex nihilo, ou à partir d'une application préexistante, que vous n'utiliserez l'assistant Création d'applications. L'assistant donne un préprogramme tout à fait fonctionnel ; mais vous développerez avec le temps un style de programmation personnel, et vous préférerez dans la plupart des cas travailler sur la base d'une de vos créations précédentes. Le style vient avec le temps et la pratique : soyez patient et explorez Visual Basic. Faites des essais, n'ayez pas peur de vous fourvoyer, et attendez-vous à commettre des erreurs chaque fois que vous écrirez un programme. La programmation, c'est de la création. Et vous découvrirez combien, avec Visual Basic, cette création peut être plaisante.

L'assistant Création d'applications est prêt à servir dès le lancement de Visual Basic. La boîte de dialogue Nouveau projet, présentée à la Figure 1.3, s'affiche lorsque vous ouvrez Visual Basic depuis le bouton Démarrer de Windows. Les onglets de la boîte de dialogue Nouveau projet proposent les choix suivants :

- **Nouveau.** Vous permet de créer une nouvelle application à partir de rien ou à l'aide d'un assistant.
- **Existant.** Vous permet de sélectionner et d'ouvrir un *projet* Visual Basic existant.
- **Récent.** Affiche une liste des projets Visual Basic récemment ouverts ou créés.

Si vous avez refermé la boîte de dialogue Nouveau projet et que vous souhaitiez par la suite lancer l'assistant Création d'applications, choisissez la commande Nouveau projet du menu Fichier pour afficher de nouveau la boîte de dialogue. Cette fois, par contre, les onglets Existant et Récent n'apparaissent pas, car la commande de menu implique que vous souhaitiez partir sur un nouveau projet.

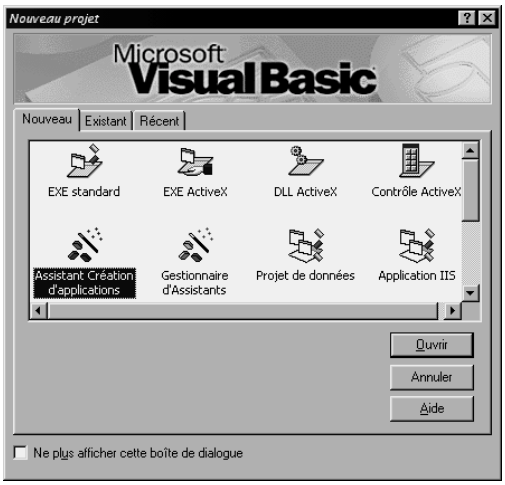


Figure 1.3
L'assistant Création d'applications peut être sélectionné depuis la boîte de dialogue Nouveau projet.

Définition

Le projet est la somme des fichiers qui constituent votre application. Une seule application peut être composée de plusieurs fichiers, rassemblés sous le nom de projet. Une partie de ces fichiers contient le code ; une autre partie contient des descriptions d'écrans à l'intérieur de leurs fenêtres de feuilles respectives ; une autre, enfin, contient des informations avancées qui permettront à votre programme de communiquer avec d'autres programmes et modules au sein du système d'exploitation.

Info

Faire
Pour que la boîte de dialogue Nouveau projet cesse de s'afficher automatiquement à chaque démarrage, cochez la case Ignorer cette boîte de dialogue à l'avenir. Au prochain démarrage de Visual Basic, elle n'apparaîtra pas.

L'assistant se lance lorsque vous double-cliquez sur l'icône Assistant Création d'applications. Le premier écran qui s'affiche est un écran d'introduction qui explique que l'assistant est prêt à commencer. (Cet écran vous permet également de charger un profil définissant des options particulières ; nous n'y aurons pas recours dans cet ouvrage.) Comme avec la plupart des assistants, lorsque vous avez terminé votre lecture et vos sélections sur un écran, vous cliquez sur le bouton Suivant pour passer à l'écran suivant. La Figure 1.4 montre la page suivante de l'assistant, dans laquelle vous devez choisir un type d'interface.

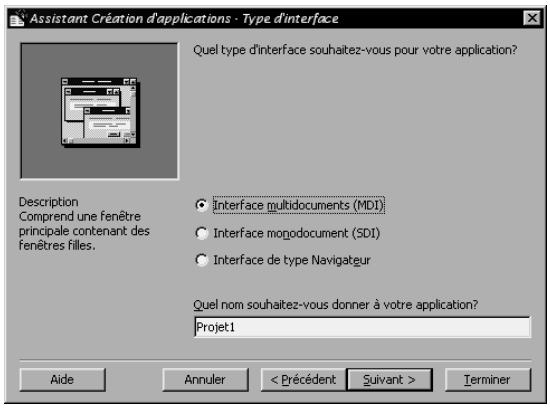


Figure 1.4
Le type d'interface détermine la façon dont votre application manipulera les fenêtres multiples.

Voici les options qui vous sont proposées :

- **Interface multidocument (MDI).** Permet à votre application de contenir plusieurs fenêtres de documents. Avec une telle interface, vous travaillez sur plusieurs jeux de données dans plusieurs fenêtres à l'intérieur du programme. Chaque fenêtre de document est appelée *fenêtre fille*.
- **Interface monoducument (SDI).** Une seule fenêtre peut être ouverte à la fois dans l'application. Vos applications seront, dans la plupart des cas, des applications SDI.
- **Style de l'explorateur.** Permet à votre application d'exploiter une interface de type Explorateur Windows, avec dans le panneau de gauche un sommaire des rubriques et dans le panneau de droite le détail de la rubrique choisie.

Cliquez sur l'une de ces options pour obtenir une description et afficher un schéma en réduction d'une fenêtre de programme type. Vous emploierez souvent l'interface monodocument, car la plupart des applications ne nécessitent qu'une seule fenêtre de données ouverte à la fois. Pour ce premier exemple, sélectionnez l'option Interface monodocument.

L'écran de l'assistant vous demande aussi le nom de votre projet. Le nom par défaut, Projet 1, laisse un peu à désirer, aussi le renommerez-vous en PremièreApp (les espaces sont interdites). Cliquez ensuite sur Suivant pour afficher la prochaine fenêtre de l'assistant, telle qu'elle est reproduite en Figure 1.5.

L'assistant Création d'applications ajoute au menu de l'application les options que vous avez sélectionnées. Il s'agit des options classiques que l'on retrouve dans la plupart des applications Windows. Les menus eux-mêmes seront du type menu déroulant standard. Vous avez le choix entre diverses options pour la barre de menus (Fichier, Edition, etc.) ainsi qu'entre des options de sous-menus, telles que Nouveau, Ouvrir ou Fermer.

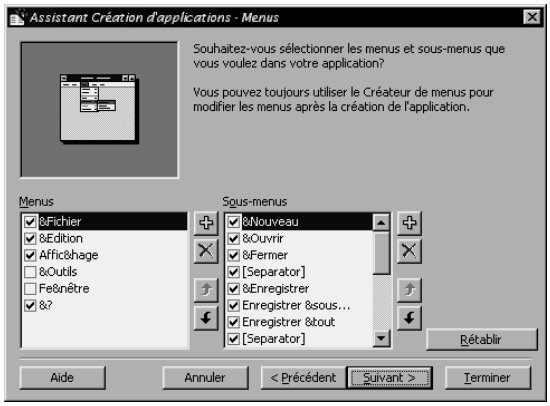


Figure 1.5
Sélectionnez les options à inclure dans le menu de votre application.

L'esperluette (&) précédant la lettre d'un nom de menu indique la touche de raccourci, qui apparaîtra en souligné ; par exemple, &Nouveau indique que Nouveau (voyez le soulignement) apparaîtra dans le menu et que l'utilisateur pourra accéder à cette commande en appuyant sur Alt-N. Pour réellement placer le caractère esperluette dans le nom, placez-en deux ; ainsi, le nom de menu **S&&SM** donnera S&SM. Pour cette application, laissez toutes les options telles quelles (&Fichier, &Edition, Affic&hage, Fe&nêtre et &? doivent être cochées). Cliquez sur Suivant pour poursuivre.

Une fois que l'assistant Création d'applications aura terminé de générer l'application, les options de menu fonctionneront comme prévu. Par exemple, le menu Fichier se déroulera lorsque vous cliquerez sur Fichier ou appuierez sur Alt-F.

Sur l'écran suivant de l'assistant, reproduit en Figure 1.6, vous choisissez les boutons de barre d'outils que contiendra votre application. Comme vous pouvez le constater, l'assistant Création d'applications fait déjà une bonne partie du travail. En créant la barre d'outils initiale, l'assistant se charge d'une corvée qui, autrement, vous aurait échue. Le panneau de gauche présente les boutons disponibles, tandis que le panneau de droite affiche les boutons (et les séparateurs qui les espacent) présélectionnés pour l'application. Comme pour les options de menus de l'écran suivant, nous accepterons tous les paramètres par défaut. Cliquez sur Suivant.

L'assistant affiche maintenant l'écran Ressources, dans lequel vous sélectionnez les ressources qui seront exploitées par votre programme, tels que des fichiers texte multilangages. Les programmes simples ne requièrent généralement pas de ressources externes. Pour cet exemple, maintenez l'option par défaut Non. Cliquez sur Suivant pour continuer.

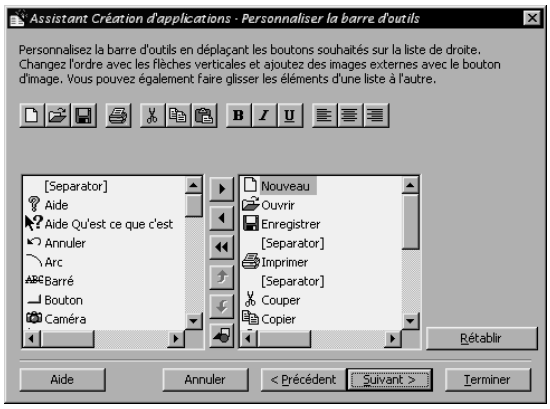


Figure 1.6
En créant la barre d'outils initiale, l'assistant Création d'applications vous fait gagner du temps.

L'écran qui s'affiche, Connexion à Internet, vous permet d'intégrer à votre application une interface Internet. Si vous sélectionniez l'option Oui dans cette fenêtre (ce que vous ne ferez pas pour l'instant), l'assistant Création d'applications ajouterait à l'application un navigateur Internet complet, qui fonctionnerait en gros comme Internet Explorer. Vos applications peuvent être reliées à l'Internet sans que vous ayez à programmer quoi que ce soit. Lorsque l'utilisateur entre une adresse Internet (ou URL, pour *Uniform Resource Locator* — localisateur unifié de ressources), comme **http://www.ssm.fr**, le navigateur affiche la page Web correspondante dans la fenêtre de navigation de l'application, utilisant pour se connecter le service Internet par défaut du PC. Une page de démarrage par défaut peut être spécifiée, qui s'affichera automatiquement dès que l'utilisateur lance le navigateur.

L'intégration d'un navigateur Web à votre application suppose que l'utilisateur dispose d'une connexion Internet. Si tel n'est pas le cas, une erreur sera générée lorsque l'utilisateur tentera de lancer le navigateur.

Cette première application ne nécessitant pas d'accès à l'Internet, nous ne modifierons pas les sélections par défaut de cet écran. Cliquez sur Suivant. La fenêtre qui s'affiche vous propose d'intégrer à votre application les écrans standards suivants :

- **Ecran de présentation au démarrage.** Portant le nom de l'application, cet écran s'affiche chaque fois que le programme est lancé.
- **Boîte de dialogue de connexion.** Boîte de dialogue dans laquelle l'utilisateur entre son nom et son mot de passe ; peut participer à la sécurité de votre application.

- **Boîte de dialogue de paramétrage d'options.** Boîte de dialogue à onglets, vierge par défaut, dans laquelle les utilisateurs spécifient des attributs que vous avez définis pour l'application.
- **Boîte de dialogue A propos de.** S'affiche lorsque l'utilisateur choisit l'option A propos de du menu "?" de l'application.

Pour les besoins de notre application, cliquez sur l'option Boîte de dialogue A propos de.

Le bouton Modèles de feuilles vous permet de choisir entre plusieurs modèles, disponibles depuis le dossier Modèles de Visual Basic. Les modèles sélectionnés sont intégrés à l'application. Le modèle Addin (complément) permet d'ajouter une feuille de votre propre librairie. Le modèle Odbc log in (ouverture de session ODBC) offre aux utilisateurs un accès aux bases de données avancées. Le modèle Tip of the day (Astuce du jour) affiche de façon aléatoire une rubrique conseil à chaque démarrage de l'application.

Un modèle de feuille est une feuille prédéfinie, mais que vous pouvez modifier selon vos besoins. Les modèles de feuilles proposent les fonctionnalités susceptibles d'être utilisées dans différentes applications.

Après avoir sélectionné la feuille standard Boîte de dialogue A propos de, vous pouvez cliquer sur Suivant pour jeter un coup d'œil à l'écran Feuille d'accès aux données, qui vous permet de rattacher à l'application des fichiers externes de bases de données. Nous n'utiliserons pas cette fonction pour l'instant ; vous pouvez donc cliquer sur le bouton Terminer afin que Visual Basic achève la création de l'application initiale.

En cliquant sur le bouton Afficher le rapport, vous faites apparaître un récapitulatif du projet qui vient d'être conçu ; le rapport indique également les modifications qui peuvent être apportées, et vous renvoie à d'autres assistants utiles.

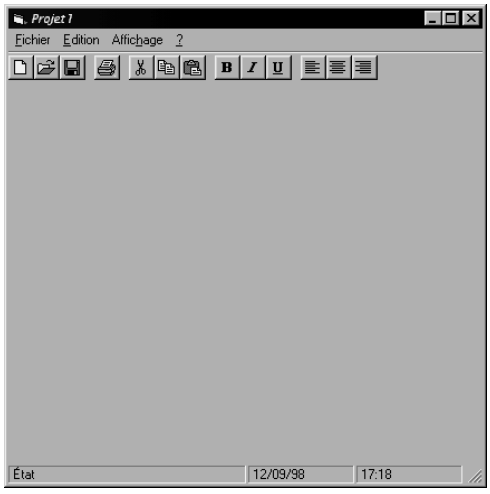
Bravo ! Vous venez de créer votre première application, sans connaître grand-chose de Visual Basic et sans *rien* connaître du langage de programmation Visual Basic. Au bout de quelques instants, Visual Basic vous fait savoir que la création de votre application est terminée. Cliquez sur OK pour faire disparaître la boîte de dialogue. Vous pouvez maintenant lancer votre application.

Après avoir chargé une application depuis le disque ou après en avoir créé une, il convient de l'exécuter. Vous la verrez ainsi "tourner", comme le verront les utilisateurs une fois que vous aurez achevé les tests et la compilation. Visual Basic, c'est un peu comme une cuisine. Vous êtes le cuisinier ; votre applica-

tion, la recette. Changez la recette (l'application), et le plat (le programme résultant) sera différent. Selon la complexité de l'application prévue, la phase de programmation peut être assez longue ; cela, même si vous avez recours à l'assistant Création d'applications pour générer le programme initial. Pour voir "tourner" le programme alors que vous le créez, il faut l'exécuter.

Pour lancer le programme, choisissez Exécution, Exécuter. (L'exécution en mode interactif est l'option par défaut.) Vous pouvez voir dans le menu la touche de raccourci correspondante, F5. La Figure 1.7 montre la fenêtre qui s'affiche.

Figure 1.7
Votre première application est créée !



Avec l'assistant Création d'applications, vous avez produit une programme tout à fait fonctionnel (bien que squelettique et limité), simplement en répondant à des questions. La nouvelle application peut être décrite comme suit :

- Une fenêtre de programme standard s'affiche, qui peut être redimensionnée et déplacée. Le titre du projet, PremièreApp, apparaît dans la barre de titre de la fenêtre.
- Une barre d'état affiche la date et l'heure. Cette barre d'état peut être désactivée depuis le menu Affichage.
- Un menu de travail propose quatre options. Seule l'option A propos du menu "?" fonctionne (essayez-la) ; mais les options de menu classiques, telles que Fichier, Ouvrir (qui ouvre une boîte de dialogue de localisation de fichier) ou Edition, Couper, sont déjà là à attendre que vous leur affectiez du code actif. Suivant en cela la convention Windows, la boîte de dialogue A propos de propose un bouton Infos système.

La fenêtre Infos système affiche un récapitulatif complet du système d'exploitation et du matériel de l'utilisateur. Visual Basic génère ce récapitulatif en exécutant un programme spécifique qui détermine la configuration exacte de la machine. (Ce programme Infos système peut être appelé depuis d'autres emplacements que la boîte de dialogue A propos de.) Un tel récapitulatif peut se révéler très utile lorsque les utilisateurs vous exposent un problème lié à l'une de vos applications. Il suffit de demander à l'utilisateur d'afficher pour vérifier que son système d'exploitation et son matériel répondent bien aux exigences du programme. La fenêtre Infos système, en outre, permet de consulter rapidement l'état des ressources, telles que l'espace disque et la mémoire disponibles.

- Une barre d'outils standard est affichée, à laquelle vous pouvez adjoindre de nouvelles fonctionnalités, et qui peut être activée et désactivée depuis le menu Affichage.

Cette application ne fait pas grand-chose, mais elle est fonctionnelle et ne demande plus de votre part qu'un travail de finition. Vous pouvez facilement modifier et enrichir l'application, ses menus, ses fenêtres. Pour l'instant, l'application n'est que le réceptacle de fonctionnalités à venir. Cependant l'assistant Création d'applications, en générant ce canevas encore grossier, vous a épargné bien de la peine. Comme vous le verrez dans la leçon de demain, il est assez facile de créer de toutes pièces un projet fonctionnel ; mais l'assistant Création d'applications jette les bases requises par la plupart des applications. Alors pourquoi s'en priver ?

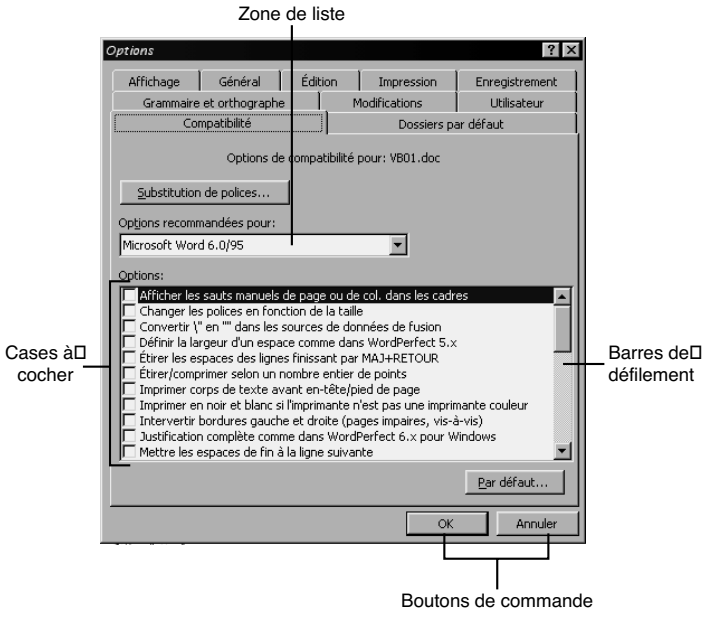
Pour quitter l'application en cours d'exécution, choisissez Fichier, Quitter. Visual Basic demande si vous souhaitez enregistrer le projet ; répondez non. Inutile d'enregistrer ce début d'application, puisqu'il suffit de lancer l'assistant Création d'applications de nouveau pour revenir au même point.

La programmation événementielle

La Figure 1.8 montre une fenêtre de programme Windows. Cette fenêtre contient plusieurs types de contrôles Windows, tels que boutons de commande, cases à cocher et barres de défilement. Ces contrôles forment seulement un exemple des multiples contrôles Windows que l'environnement de programmation Visual Basic met à votre disposition.

De par sa nature visuelle, Visual Basic exige ce type de contrôles. En effet, et contrairement aux programmes écrits en langages textuels "à l'ancienne", les programmes Windows réagissent à des *événements*. Un événement peut être provoqué par l'un de ces contrôles, mais aussi ressortir aux activités internes, comme l'horloge du PC. Les événements se produisent toujours selon un ordre aléatoire.

Figure 1.8
Les programmes Windows réagissent à des événements.



Par exemple, l'utilisateur du programme de la Figure 1.8 pourrait cliquer sur l'un des boutons, cocher l'une des cases ou activer l'une des zones de liste déroulante. Cet utilisateur peut procéder à des manipulations différentes, dans un ordre différent, chaque fois qu'il se sert du programme. Pour répondre efficacement aux actions de l'utilisateur et aux autres activités propres à déclencher un événement, on emploie des techniques de *programmation événementielle*.

Par événement, on désigne toute action qui se déclenche lors de l'exécution du programme : clic de souris, frappe au clavier, etc. Est orienté événement un programme qui réagit aux événements Windows.

Votre programme doit gérer les événements aléatoires. Mais, dans le cas où plusieurs programmes Windows s'exécutent en même temps, chacun doit être capable d'analyser et de répondre aux événements qui le concernent.

Comme le montre la Figure 1.9, Windows gère quelques événements, mais passe le plus souvent la main aux programmes en cours d'exécution. Windows étant un système d'exploitation multitâche, plusieurs programmes peuvent être exécutés simultanément. Votre programme doit traiter chacun des événements appropriés à mesure qu'ils se

produisent, tout en ignorant ceux qui ne le concernent pas. Par exemple, un programme censé afficher un message d'avertissement à intervalles réguliers doit vérifier auprès de l'événement horloge le laps de temps qui s'est écoulé depuis le dernier message. Tout autre programme simultanément en éxécution, et qui n'a pas à vérifier l'heure, devra ignorer les messages correspondants envoyés par Windows.

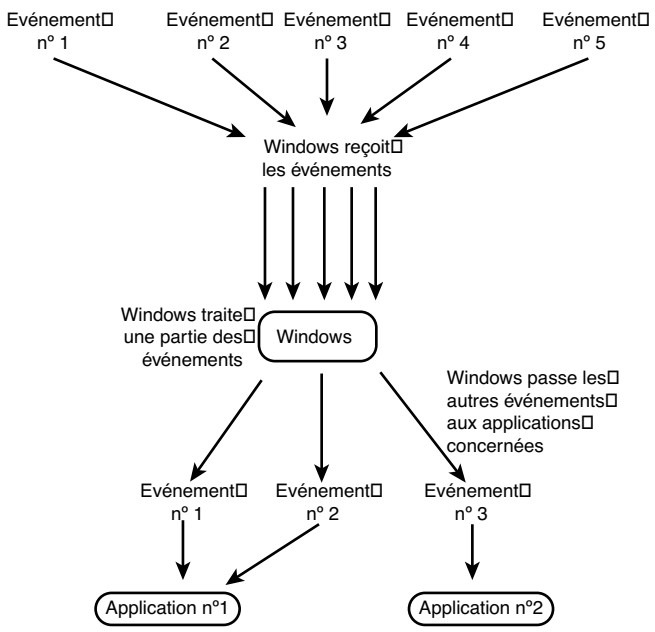


Figure 1.9
Votre programme doit réagir à certains événements en ignorant les autres.

Un programme Visual Basic est constitué de l'interface visuelle, fenêtres et contrôles, avec laquelle l'utilisateur interagit. Le code de programmation vient relier tout cela ensemble. Chaque contrôle est à la fois automatique, d'un fonctionnement normalisé, et paramétré selon le code du programme spécifique. Par exemple, un bouton de commande réagira visuellement toujours de la même manière lorsqu'un utilisateur clique dessus. Pour faire fonctionner ce bouton, vous avez juste à le placer sur la feuille (la fenêtre de programme). Notez que la plupart des boutons de commande peuvent être activés par la touche entrée aussi bien que par la souris. Sous tous les autres aspects, en revanche, le bouton est totalement sous votre contrôle : vous décidez du nom ou de l'image qui doit y apparaître, de sa taille, de sa couleur, etc. Ce sont des *propriétés* modifiables, bien que Visual Basic y assigne des valeurs par défaut. Ce sont ces propriétés qui distinguent un bouton de commande d'un autre.

Info

En tant qu'elles définissent l'apparence et le comportement des contrôles, les propriétés permettent de les différencier. Ces propriétés prennent diverses valeurs : couleur, étiquette texte, taille, emplacement sur la feuille. En disposant un contrôle sur la feuille, vous lui attribuez les propriétes qui font son "identité".

La Figure 1.10 montre une fenêtre contenant de nombreux boutons de commande. Si, pour aucun de ces boutons, le code ne spécifiait un comportement précis, ils réagiraient tous de la même manière aux clics de la souris : s'enfoncer (visuellement) et déclencher un événement Windows "clic". C'est pourquoi l'on affecte à chacun des propriétés particulières, quant à la légende, à la taille, à la couleur, etc.

Figure 1.10
Chacun des multiples contrôles est différencié par les propriétés que lui affecte le code.

Une fois que vous avez placé les contrôles sur la feuille et que vous leur avez spécifié des propriétés individuelles, vous êtes prêt à écrire le code qui répondra aux événements déclenchés par ces contrôles. Un même contrôle peut déclencher plusieurs types d'événements. Par exemple, à un même bouton de commande peut correspondre un événement clic ou un événement double-clic, selon ce que fait l'utilisateur. Vous écrivez le code pour déterminer lequel de ces événements doit être ignoré, lequel doit être géré, et comment.

Astuce

Si vous écrivez du code pour un événement particulier, le programme répondra à cet événement dès qu'il se produira dans le cours de l'exécution. Si toutefois vous n'affectez pas de code à l'événement, et qu'il se produise, votre programme ignorera l'information transmise par Windows pour cet événement.

Le code "derrière" la feuille de programme ne fonctionne pas comme un long listing textuel, mais plutôt comme une série de courtes sections de code, chacune chargée de répondre aux événements des contrôles de la feuille. Chacune de ces sections se tourne les pouces jusqu'à ce que "son" événement se produise ; lorsque l'événement se produit, le programme exécute le code correspondant. Par exemple, pour qu'un clic du bouton droit sur un *objet* (bouton de commande, etc.) déclenche l'émission d'un bip et l'affichage d'un avertissement, vous devez écrire le code pour ce bip et ce message. Et le code en question ne sera exécuté que si l'utilisateur clique du bouton droit sur l'objet.

Un objet est un élément de programme Visual Basic, tel qu'un contrôle, une feuille ou un module de code contenant des instructions.

Comment tous ces fragments fonctionnent-ils ensemble ? La réponse viendra, disons, dans une vingtaine de chapitres. Pour la leçon de demain, vous commencerez par apprendre à spécifier les propriétés des contrôles, ainsi qu'à gérer ces contrôles lorsque vous créez votre toute première application de zéro, sans l'aide de l'assistant Création d'applications. La théorie seule n'est pas suffisante — il faut se mettre au clavier et commencer à placer les contrôles, à en définir les propriétés, et à écrire le code des événements correspondants.

En résumé

Vous voilà bien en route vers la maîtrise de Visual Basic. Vous avez appris dans ce chapitre les bases de la programmation. Une fois assimilé le processus de programmation, vous êtes équipé pour commencer l'utilisation de Visual Basic, l'un des environnements de programmation les plus performants d'aujourd'hui.

Cette leçon vous a enseigné à concevoir et à écrire un programme. Visual Basic a bouleversé les méthodes de conception des programmeurs, par la facilité avec laquelle on peut prototyper le concept du programme et passer de ce prototype au produit fini. En programmation, il faut toujours revenir sur son ouvrage. Il est rare qu'un programme fonctionne parfaitement du premier coup ; mais, comme nous l'avons vu, l'environnement interactif de Visual Basic vous épargne beaucoup de travail — et donc beaucoup d'erreurs.

L'assistant Création d'applications génère un squelette de programme que vous enrichissez par la suite pour en faire une application autonome, fonctionnelle et conforme à vos besoins. Enrichir signifie ajouter des contrôles, définir leurs propriétés, et écrire le code qui permettra au programme de réagir et d'interagir correctement avec ces contrôles. Dans les chapitres qui suivent, vous apprendrez à maîtriser ces détails pour aboutir à un programme efficace.

Questions-réponses

Q Faut-il suivre les étapes du processus de programmation (conception, création des éléments visuels, etc.) pour tous les programmes Visual Basic, ou seulement pour les petits programmes ?

R Plus le programme est important, plus il est important de suivre cette procédure ; et le programme se complexifie rapidement, à mesure que vous l'enrichissez de nouvelles fonctionnalités. Telle fonctionnalité peut affecter telle autre ; aussi, plus vous êtes prévoyant, plus vous planifiez, et moins vous aurez à retravailler et à corriger par la suite. Heureusement, l'environnement Visual Basic facilite la modification des programmes, même quand des changements majeurs dans la structure sont impliqués. Quand vous utilisez l'assistant Création d'applications, bien sûr, la conception est la deuxième étape. A mesure que vous progresserez dans ce livre, vous apprendrez à écrire et à concevoir efficacement vos programmes.

Q L'assistant Création d'applications génère-t-il du code ?

R L'assistant Création d'applications génère du code, mais pas beaucoup. Chaque instruction a pour but de faire exécuter au programme une tâche spécifique : effectuer un calcul, etc. En tant que programmeur, c'est votre boulot que de produire le code spécifique.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Sur quel langage Microsoft s'est-il fondé pour élaborer Visual Basic ?
2. Pourquoi Visual Basic est-il adapté aux débutants comme aux programmeurs confirmés ?
3. Qu'est-ce qui est le plus important pour les novices en Visual Basic : le langage de programmation ou l'interface visuelle ?
4. Quelle est la différence entre une *fenêtre de feuille* est une *fenêtre d'application* ?

5. Que signifient les termes *bogue* et *déboguer* ?
6. Qu'est-ce qui s'exécute le plus vite : un programme écrit dans un langage interprété ou un programme écrit dans un langage compilé ?
7. Qu'est-ce qui est plus facile à déboguer : un programme écrit dans un langage nteprété ou un programme écrit dans un langage compilé ?
8. Quelle est la différence entre un Ecran de présentation au démarrage et un écran Astuce du jour ?
9. Quelle est la différence entre un contrôle et la valeur de propriété d'un contrôle ?
10. Les contrôles contiennent le code qui leur permet de réagir aux actions de l'utilisateur. Vrai ou faux ?

Exercice

En vous servant de l'assistant Création d'applications, créez une application qui inclut, en plus des diverses options que vous avez choisies pour l'exemple de ce chapitre, un navigateur Internet et un Ecran de présentation au démarrage. Exécutez l'application pour voir le fonctionnement de l'accès Internet. Si vous ne disposez pas d'une connexion Internet, vous obtiendrez un message d'erreur en essayant de lancer la fenêtre de navigation. Mais créez le projet quand même, pour la pratique.

Chapitre 2

L'environnement et les outils Visual Basic

Maintenant que vous avez pu voir combien l'assistant Création d'applications est simple d'utilisation, vous êtes prêt à faire le plongeon : créer un programme à partir de zéro. Il n'est pas très difficile de créer une application sans l'aide de l'assistant Création d'applications, mais vous devez néanmoins comprendre l'environnement Visual Basic avant de vous atteler à l'écriture. Au terme de ce chapitre, vous devriez être capable de vous repérer dans l'environnement Visual Basic pour créer vos programmes comme un pro : "à la main".

Voici ce que nous étudierons aujourd'hui :

- les éléments de l'environnement Visual Basic ;
- le placement des contrôles sur une feuille ;
- l'enregistrement du projet et des fichiers associés ;
- la fenêtre Propriétés et ses composants ;
- l'accès à la fenêtre Code.

L' environnement Visual Basic

Au cours de ces vingt et un jours d'apprentissage, c'est dans l'environnement Visual Basic que vous étudierez et construirez des programmes. Plus vite vous vous familiariserez avec cet environnement, dont le principe réside essentiellement dans le jeu des diverses fenêtres, plus vite vous maîtriserez la programmation Visual Basic. La Figure 2.1 présente un écran Visual Basic dont les principaux composants sont désignés.

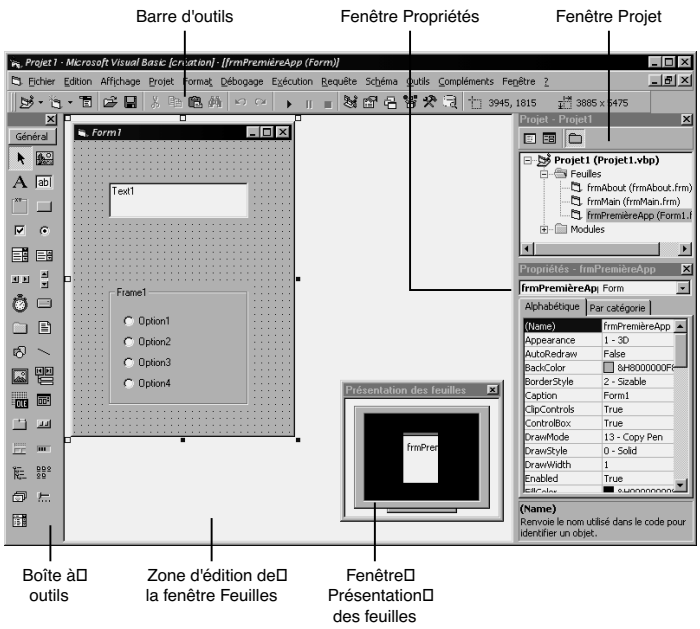


Figure 2.1
Vous devez assimiler le fonctionnement des composants Visual Basic.

La fenêtre Nouveau projet

Comme nous l'avons vu lors de la leçon précédente, la fenêtre Nouveau projet s'affiche au lancement de Visual Basic ou lorsque vous choisissez la commande Fichier, Nouveau projet. Dans le cadre de votre apprentissage, c'est depuis la fenêtre Nouveau projet que vous commencerez la plupart de vos applications.

Quand vous ne vous servirez pas de l'assistant Création d'applications pour définir une structure d'application, comme nous l'avons fait dans la leçon d'hier, c'est l'icône EXE standard que vous choisirez pour créer un programme autonome. Cette icône est nommée ainsi d'après l'extension que portera le programme une fois compilé : `.EXE`, pour *exécu-*

table. Même si vous ne compilerez pas tout de suite vos applications, c'est l'icône EXE standard que vous sélectionnerez le plus souvent dans votre apprentissage de Visual Basic.

Une application EXE standard est une application qui peut être compilée ou exécutée de façon interprétée.

Vous verrez dans la fenêtre Nouveau projet de nombreuses icônes portant le nom d'ActiveX. ActiveX est le nom donné aux contrôles que vous créez vous-même. Ces contrôles portent l'extension de noms de fichiers `.OCX`*, et peuvent être intégrés à l'environnement Visual Basic de sorte qu'ils demeurent dans votre fenêtre Boîte à outils. Vous pouvez écrire des applications qui deviendront des contrôles et pourront ainsi servir à des projets ultérieurs. En fait, ActiveX est une désignation assez large qui s'applique également à d'autres domaines informatiques.*

Souvenez-vous que Visual Basic n'est rien d'autre qu'un programme Windows, certes vaste, qui vous permet de créer de nouveaux programmes Windows. Les barres d'outils, menus, boîtes de dialogue et fenêtres de l'environnement Visual Basic fonctionnent d'une façon tout à fait semblable aux autres applications Windows. Vous ne devriez donc pas avoir de mal à vous y familiariser.

La barre d'outils

Selon votre utilisation de Visual Basic, la barre d'outils située sous la barre de menus se modifie. En tout, quatre barres d'outils sont disponibles :

- **Débogage.** S'affiche lorsque vous employez les outils de débogage pour corriger votre programme.
- **Edition.** Facilite l'édition du code Visual Basic.
- **Editeur de code de feuille.** Aide à la disposition des objets sur la feuille.
- **Standard.** Barre d'outils par défaut, affichée sous la barre de menus.

L'affichage de ces barres d'outils peut être activé et désactivé depuis le menu Affichage, Barres d'outils. Chaque barre d'outils présente une multitude de boutons, grâce auxquels vous pouvez accéder aux fonctionnalités courantes sans passer par une succession de commandes de menus. Dans le cours de vos développements Visual Basic, vous rencontrerez de nombreux boutons très utiles, alors que d'autres ne vous seront jamais d'aucun usage. Nous signalerons, dans ces leçons, des boutons de barres d'outils

susceptibles d'accélérer le développement de vos programmes, mais nous ne saurions en aucun cas fournir une référence exhaustive des boutons disponibles, tous n'étant pas d'une égale utilité pour notre propos.

Faire
Quand vous ne reconnaissez pas un bouton, maintenez-y un instant le curseur de la souris : une info-bulle s'affiche, qui indique la fonction du bouton.

Ne pas faire
N'essayez pas de mémoriser tous les boutons de toutes les barres d'outils.

Chaque barre d'outils peut être ancrée ou désancrée. C'est-à-dire que vous pouvez faire glisser une barre d'outils depuis sa position, sous la barre de menus, pour former une barre d'outils flottante. Il est ainsi possible de placer une barre d'outils près de l'élément auquel elle s'applique, afin que les boutons requis soient rapidement accessibles. Libre à vous de replacer, par la suite, la barre d'outils à sa position initiale, en la faisant glisser sous la barre de menus, où elle restera jusqu'au prochain changement.

La Boîte à outils

La fenêtre Boîte à outils n'est pas une barre d'outils. Comme son nom l'indique, il s'agit d'une collection d'outils grâce auxquels vous disposez les contrôles sur la feuille. En progressant dans ces leçons, vous apprendrez à ajouter et à retirer les boutons de la Boîte à outils. La Figure 2.2 montre un jeu des outils les plus communs.

Avec la Boîte à outils, vous ne serez jamais à court d'outils. Si votre application requiert plus d'un bouton de commande, l'outil CommandButton de la Boîte à outils vous les fournira. Les boutons de la Boîte à outils génèrent les outils à votre feuille à mesure que vous les demandez. C'est ce que nous verrons à la dernière section de cette leçon, où nous créerons une application ex nihila.

La fenêtre Feuille

C'est dans la fenêtre Feuille que vous effectuerez la plupart de vos opérations. Toutes les feuilles de vos programmes, qui formeront l'arrière-plan visible de l'application, seront créées dans la zone centrale d'édition où s'affiche la fenêtre Feuille. Vous pouvez ajuster la fenêtre Feuille de façon à donner aux fenêtres créées à l'intérieur les dimensions voulues. (Les barres de défilement vous permettent de faire défiler le contenu de la fenêtre Feuille pour qu'apparaissent les éléments masqués.)

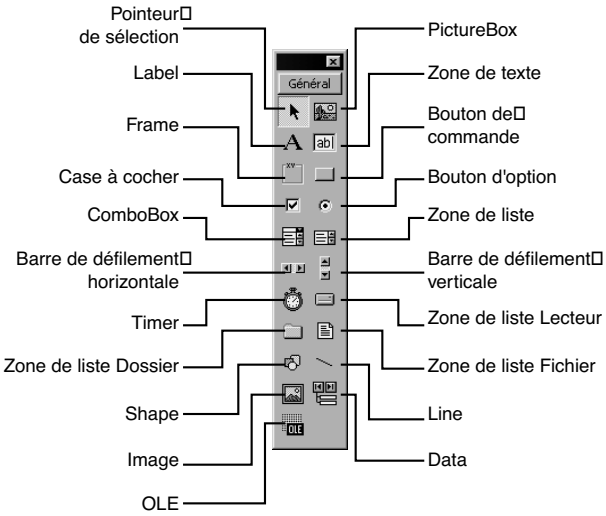


Figure 2.2
La Boîte à outils contient les outils qui, sur votre feuille, deviennent des contrôles.

Gardez à l'esprit qu'une application contient une multitude de feuilles. Chacune de ces feuilles peut être affichée dans sa propre zone d'édition de la fenêtre Feuille ; c'est ce que montre la Figure 2.3. La feuille active est celle dont la barre de titre est en surbrillance. Pour activer une feuille, on clique n'importe où à l'intérieur ou sur la barre de titre.

La fenêtre Présentation des feuilles

La fenêtre Présentation des feuilles est une petite fenêtre intimement liée à la fenêtre Feuille. Cette fenêtre a cela de très utile, qu'elle donne un aperçu de la disposition des feuilles. Quand la fenêtre Feuille contient plusieurs feuilles, la fenêtre Présentation des feuilles affiche un schéma miniature de chacune de ces feuilles. La fenêtre vous permet de visualiser l'organisation des feuilles sur l'écran, tel que l'utilisateur le verra, ainsi que leur évolution au gré de l'utilisation du programme.

La fenêtre Présentation des feuilles ne se contente pas de donner un aperçu du positionnement des feuilles sur l'écran final. Elle vous permet également de déplacer ces feuilles en les faisant simplement glisser vers une nouvelle position. Si, par exemple, vous souhaitez qu'une feuille apparaisse au centre de l'écran, il suffit de la faire glisser là, dans la fenêtre Présentation des feuilles ; et c'est ce que l'utilisateur verra lorsqu'il exécutera le programme.

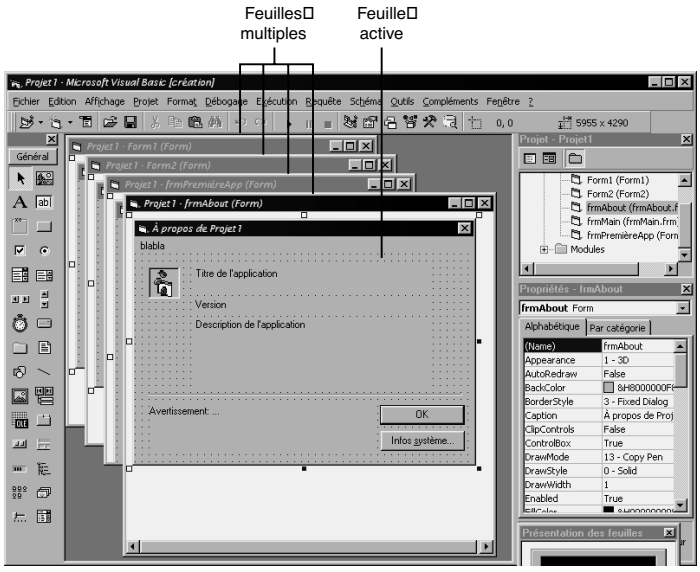


Figure 2.3
Les zones d'édition de la fenêtre Feuille vous permettent de travailler sur plusieurs feuilles à la fois.

Info

Lorsque vous en saurez assez sur le langage de programmation Visual Basic, vous serez en mesure de spécifier dans le code seul la position exacte d'une feuille sur l'écran. Vous pouvez également indiquer à Visual Basic de centrer automatiquement la feuille dès que la forme est créée, et indépendamment de ce qu'affiche la fenêtre Présentation des feuilles au cours du développement.

Astuce

Beaucoup de programmeurs ferment la fenêtre Présentation des feuilles afin de libérer de la place pour d'autres fenêtres.

La fenêtre Projet

Dans la fenêtre Projet, vous gérez les composants de votre application. Comme le montre la Figure 2.4, la fenêtre Projet peut devenir passablement encombrée. Un programme Windows, que nous devrions appeler, comme l'a enseigné la leçon précédente, une application, peut recouvrir plusieurs fichiers. Avant que le programme ne soit compilé, les fichiers liés à Visual Basic peuvent devenir encore plus nombreux. La fenêtre Projet vous permet de gérer tous ces composants et d'amener dans la zone d'édition celui sur lequel vous souhaitez travailler.

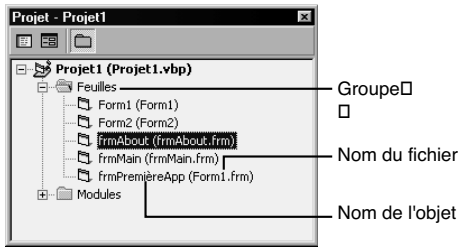


Figure 2.4
La fenêtre Projet contient l'ensemble des composants de votre projet.

Info

La fenêtre Projet est également appelée Explorateur de projets, en raison de son interface semblable à celle de l'explorateur Windows, et dans laquelle on peut étendre et réduire les groupes d'objets.

Dans la fenêtre Projet, la liste des composants est présentée comme une structure arborescente. Les objets corrélés sont affichés ensemble. Les signes moins (–) et plus (+) vis-à-vis des groupes d'objets permettent respectivement de réduire ou d'étendre l'affichage des détails. Si, par exemple, vous cliquez sur le signe plus de l'objet Feuilles, la liste des projets en cours s'affiche. Quand vous double-cliquez sur l'une des feuilles, la fenêtre correspondante s'affiche dans la zone d'édition de la fenêtre Feuille.

Chaque élément de la fenêtre Projet porte à la fois un nom de projet et un nom de fichier. En Visual Basic, on assigne des noms aux objets tels que feuilles ou modules. Chaque élément est aussi enregistré sur le disque dans un fichier séparé. Le nom de fichier qui, pour un même élément, diffère du nom de projet (par exemple, seuls les noms de fichiers portent une extension) apparaît entre parenthèses à côté de l'élément. La fenêtre Projet affiche donc le nom de fichier et le nom de projet de chacun des fichiers de votre projet, et il suffit de double-cliquer sur un objet pour l'activer.

Astuce

La fenêtre Projet inclut une barre d'outils qui n'affiche que trois boutons. Le bouton Fenêtre Code affiche la fenêtre Code pour l'objet sélectionné, de sorte que vous puissiez en modifier le code. (La fenêtre Code n'apparaissait pas sur la Figure 2.1 ; elle vous sera présentée à la dernière section de cette leçon, alors que vous ajouterez du code à l'application.) Le bouton Afficher l'objet affiche la fenêtre d'objet pour l'objet sélectionné. Beaucoup d'objets se voient associer à la fois une fenêtre d'objet et une fenêtre Code. A chaque feuille, par exemple, correspondent un module de code et une fenêtre Feuille. Les boutons Fenêtre Code et Afficher l'objet vous permettent donc de basculer du code d'un objet à ses éléments visuels. Le bouton Basculer les dossiers groupe et dégroupe les éléments de la fenêtre Projet selon une interface de type Explorateur.

Voici les types d'objets qui apparaissent dans la fenêtre Projet :

- **Projets.** Une application peut être constituée de plusieurs projets, notamment lorsque vous employez des contrôles ActiveX. Les projets portent toujours l'extension `.VBP`.
- **Feuilles.** La fenêtre projet affiche la liste des feuilles de votre projet. Les feuilles portent toujours l'extension `.FRM`.
- **Modules.** Les modules de votre projet contiennent des routines (ensembles d'instructions Visual Basic) générales et réutilisables. Un module peut ainsi être utilisé par plusieurs programmes. Les modules portent toujours l'extension `.BAS`.
- **Modules de classes.** Les modules de classes sont des modules spéciaux qui définissent les objets conçus pour un projet. Les modules de classes portent toujours l'extension `.CLS`.
- **Contrôles utilisateur.** Les contrôles utilisateur sont les contrôles ActiveX que vous avez ajoutés au projet. Les fichiers des contrôles ActiveX portent toujours l'extension `.OCX`.
- **Documents utilisateur.** Les documents utilisateur sont des objets document qui décrivent les parties de votre projet. Les fichiers de documents utilisateur portent toujours l'extension `.DOB`.
- **Page de propriétés.** Les pages de propriétés (comme celles que l'on trouve dans les boîtes de dialogue à onglets) apparaissent dans un fichier de projet pour décrire un contrôle particulier. Les fichiers de pages de propriétés portent toujours l'extension `.PAG`.

D'autres éléments apparaissent parfois dans la fenêtre Projet, tels que les ressources et autres documentations que vous adjoignez au projet.

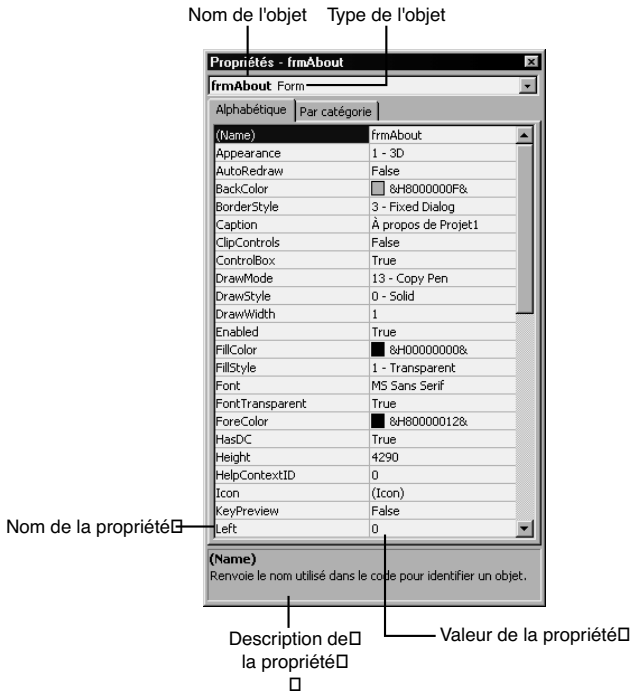
Pour la plus grande partie du développement d'applications Visual Basic, et notamment lors de ces vingt et un premiers jours, vous travaillerez uniquement sur les feuilles et les modules.

La fenêtre Propriétés

Une feuille peut contenir plusieurs contrôles. A mesure que vous en ajoutez, vous pouvez sélectionner les contrôles, simplement en cliquant dessus. Lorsqu'un contrôle est sélectionné, la fenêtre Propriétés affiche toutes les propriétés qui lui sont liées. Comme vous le verrez dans la dernière section de cette leçon, Visual Basic définit automatiquement les valeurs de propriétés initiales du contrôle dès que vous l'ajoutez. En affichant la fenêtre Propriétés d'un contrôle, vous pouvez modifier ces valeurs.

La Figure 2.5 montre une fenêtre Propriétés qui affiche quelques-unes des propriétés d'un contrôle Label. Vous pouvez constater que les informations affichées dans la fenêtre Propriétés quant au nom, au type et à la description, reflètent le contrôle sélectionné. Pour affecter une valeur à une propriété, sélectionnez la propriété et entrez la nouvelle valeur. Quand vous pouvez choisir parmi plusieurs valeurs établies, une liste déroulante s'affiche.

Figure 2.5
La fenêtre Propriétés décrit chaque propriété du contrôle sélectionné.



Chaque propriété porte un nom qui la distingue ; et chaque propriété a une valeur que vous ou Visual Basic lui assignez. Par exemple, Visual Basic nomme toujours *Command1* le premier bouton de commande que vous ajoutez à un projet. La propriété `Name` de ce premier bouton aura donc pour valeur `Command1`. Il convient naturellement de donner au bouton de commande un nom plus significatif afin de bien documenter l'application. Vous pouvez, par exemple, nommer `cmdReportPrint` un bouton qui déclenche l'impression d'un rapport.

A chaque nom d'objet doit être inclus un préfixe de trois lettres qui décrit la fonction de l'objet. Ainsi, lorsque vous consulterez la liste des objets, vous connaîtrez de chacun non

seulement le nom, mais aussi le type (bouton de commande, zone de texte, feuille, etc.). Le Tableau 2.1 propose une liste de préfixes couramment utilisés dans les noms d'objets Visual Basic. Vous pourrez vous y référer lorsqu'il vous faudra nommer des objets, dans les jours et les leçons qui vont suivre. Quand un projet contient plusieurs contrôles, ces noms vous aident à en déterminer le type et la fonction.

Tableau 2.1 : Préfixes courants à placer en tête des noms d'objets

Préfixe	*Type d'objet*
`cbo`	Zone de liste déroulante modifiable (*ComboBox*)
`chk`	Case à cocher
`cmd`	Bouton de commande
`dir`	Zone de liste des dossiers
`drv`	Zone de liste des lecteurs
`fil`	Zone de liste des fichiers
`fra`	Frame
`frm`	Feuille
`grd`	Grille
`hsb`	Barre de défilement horizontale
`img`	Image
`lbl`	Label
`lin`	Ligne
`lst`	Zone de liste
`mnu`	Menu
`mod`	Module
`ole`	OLE
`opt`	Bouton d'option
`pic`	Zone d'image

Tableau 2.1 : Préfixes courants à placer en tête des noms d'objets *(suite)*

Préfixe	*Type d'objet*
`res`	Ressource
`shp`	Forme
`tmr`	Timer
`txt`	Zone de texte
`typ`	Types de données définis par l'utilisateur
`vsb`	Barre de défilement verticale

Faire
N'employez pas de préfixe dans les noms de fichiers attribués aux objets. Les préfixes de noms d'objets Visual Basic doivent toujours êtres saisis en minuscules.

Ne pas faire
Souvenez-vous que vous pouvez déplacer, redimensionner et fermer chaque fenêtre Visual Basic. Selon ce que vous souhaitez voir du contenu d'une fenêtre, vous pouvez l'ajuster pour faire de la place à d'autres fenêtres.

Obtenir de l'aide

Visual Basic propose toute une série d'outils en ligne destinés à vous aider. Avant de créer votre application, comme la dernière section de ce Chapitre vous invite à le faire, vous devez vous familiariser avec les diverses options d'aide et apprendre à y accéder.

L'aide locale

Dans la plupart des cas, l'environnement Visual Basic vous offre toute l'aide nécessaire sans qu'il vous soit besoin d'aller voir ailleurs (sauf dans ce livre, bien sûr !). La première option du menu Aide, Sommaire, affiche une fenêtre d'aide Windows à base de HTML ; cet écran est reproduit en Figure 2.6. Le panneau gauche de la fenêtre liste les différents manuels en ligne que vous pouvez ouvrir et lire. Le panneau de droite propose un tour des rubriques d'aide, avec pour guide un certain Dr Gui (GUI est le sigle de *graphical user interface*, interface utilisateur graphique).

Le système d'aide Visual Basic est fondé sur Books Online, base de données de référence fournie avec les produits Microsoft plus anciens. Les CD-ROM MSDN sont nécessaires pour accéder à l'aide en ligne.

MSDN signifie Microsoft Developer's Network (réseau des développeurs Microsoft), et recouvre un ensemble d'articles en ligne, de CD-ROM et de lettres d'information, diffusés auprès des programmeurs depuis quelques années. L'aide Visual Basic fait maintenant partie de MSDN. Les écrans MSDN ne sont disponibles qu'une fois l'abonnement souscrit. Pour vous abonner à l'information en ligne MSDN, cliquez sur le menu ?, sur Sommaire, puis choisissez MSDN Online.

L'aspect du système d'aide chez vous peut être légèrement différent, selon la date de distribution de votre logiciel Visual Basic 6. Les écrans d'aide varient parfois d'une édition à l'autre des produits Microsoft.

Figure 2.6
L'aide Visual Basic est là pour vous sortir du pétrin.

La boîte de dialogue Aide propose différente options d'assistance en ligne ou immédiate :

- **Sommaire.** Cette option propose une aide organisée en livres, tels que "Documentation Visual Basic", "Outils et technologies", etc.
- **Index.** Cette option permet de rechercher de l'aide à partir d'un ensemble de mots clés indexés sur les références de l'option Sommaire.

- **Rechercher.** Cette option permet de rechercher une chaîne de caractères précise dans un article.
- **Favoris.** Cette option vous permet de stocker les rubriques d'aide que vous avez jugées particulièrement utiles.

Vous êtes-vous jamais demandé pourquoi un système de développement aussi vaste que Visual Basic n'est pas livré avec un épais et pesant manuel ? En fait, Visual Basic est bel et bien livré avec un manuel, plusieurs même ; mais il s'agit d'une documentation en ligne, comme le système d'aide MSDN. Il suffit de cliquer pour ouvrir l'un des "livres" du panneau gauche, et atteindre un chapitre et une page. Cette page s'affiche dans le panneau droit de la fenêtre d'aide.

Les écrans d'aide s'affichent dans une fenêtre indépendante de la fenêtre Visual Basic. Vous pouvez donc garder les deux ouvertes, et basculer de Visual Basic à la rubrique d'aide en appuyant sur Alt-Tab, ou en cliquant sur les boutons correspondants de la Barre de tâche.

Le système d'aide offre des références sur Visual Basic, les connexions aux bases de données, la programmation ActiveX, et d'autres questions techniques à propos desquelles vous aurez besoin d'information fiable dans le cours de vos travaux. Il faut se figurer le système d'aide comme un jeu complet d'ouvrages de référence coûteux qui, s'ils étaient livrés sous la forme papier avec Visual Basic, augmenteraient considérablement le prix du logiciel, tout en rendant malaisée la recherche de sujets spécifiques.

A propos de Microsoft Visual Basic est une autre entrée du menu Aide, qui affiche une boîte de dialogue A propos classique indiquant le numéro de version de Visual Basic, le numéro de série et le nom d'enregistrement. Lorsque vous cliquez sur le bouton Infos système, le programme analyse votre système puis affiche la boîte de dialogue reproduite en Figure 2.7. Ces informations système concernent aussi bien le matériel que le logiciel.

Au Chapitre 20, vous apprendrez à ajouter une aide en ligne à vos applications.

Le support technique

En sélectionnant l'option Support technique du menu Aide, vous affichez une boîte de dialogue qui explique comment contacter le support technique de Microsoft pour une assistance plus personnalisée. Il se peut en effet que le système d'aide en ligne ne réponde pas assez à un problème précis. Si, par exemple, vous constatez un comportement étrange de Visual Basic, révélant un bogue dans le logiciel lui-même, il faut en référer à Microsoft directement. (Pour ce type de problème, toutefois, commencez toujours par réinstaller Visual Basic afin de voir si le dysfonctionnement persiste. C'est sans doute le conseil qu'ils vous donneront, et vous gagnerez ainsi du temps.)

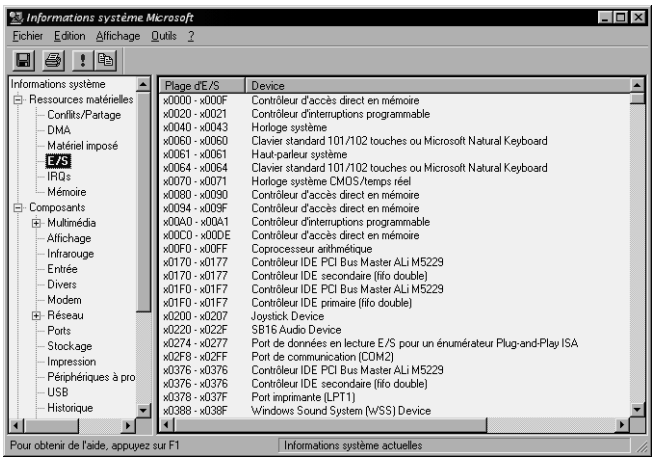


Figure 2.7
Le bouton Infos système lance une analyse de fond en comble de votre système.

Pourquoi ce type d'information est-il nécessaire pour contacter le support technique ? Après tout, tout ce dont vous avez besoin, c'est d'un numéro de téléphone et des horaires d'ouverture. Il se trouve que Microsoft offre plusieurs niveaux de support technique, du service gratuit et limité à l'abonnement annuel ; la fenêtre d'aide vous donne un résumé des options disponibles. Du reste, les coordonnées du support technique ne sont naturellement pas les mêmes d'un pays à l'autre.

L'aide en ligne

La commande Microsoft sur le Web du menu Aide vous permet de choisir entre diverses options de support en ligne. (Toutes requièrent évidemment un accès Internet.) L'option Support technique charge la page Web Microsoft dédiée à Visual Basic. Cette page, soit dit en passant, gagne à être souvent visitée, même si vous ne recherchez pas d'aide précise. Vous y trouverez un nombre appréciable d'informations de mise à jour, de corrections de bogues, d'astuces et de solutions, des exemples de code, ainsi qu'une sélection de liens actualisés. La commande Microsoft sur le Web permet également d'accéder à la page d'accueil de Microsoft, d'effectuer une recherche sur le réseau, et même d'envoyer à Microsoft des commentaires ou des conseils à propos de Visual Basic.

Visitez souvent le site Forum aux questions ; vous y trouverez une liste de réponses aux questions les plus courantes liées à la programmation Visual Basic.

Apprivoiser l'écran

Avant de terminer votre première journée d'apprentissage, vous avez créé une application Visual Basic autonome et fonctionnelle. Pour être plus précis : vous avez créé une application avec l'aide de l'assistant Création d'applications, qui a fait tout le boulot. Il s'agit maintenant d'aller plus loin et de créer une application complète à partir de zéro.

Vous comprenez mieux qu'auparavant l'environnement Visual Basic, et vous êtes en mesure de trouver de l'aide si nécessaire. Avant de suivre les étapes de la section suivante pour créer une nouvelle application, prenez le temps de charger une application existante depuis l'un des fichiers exemples livrés avec Visual Basic. Vous pourez ainsi vous familiariser avec les fenêtres qui s'affichent. Procédez comme suit :

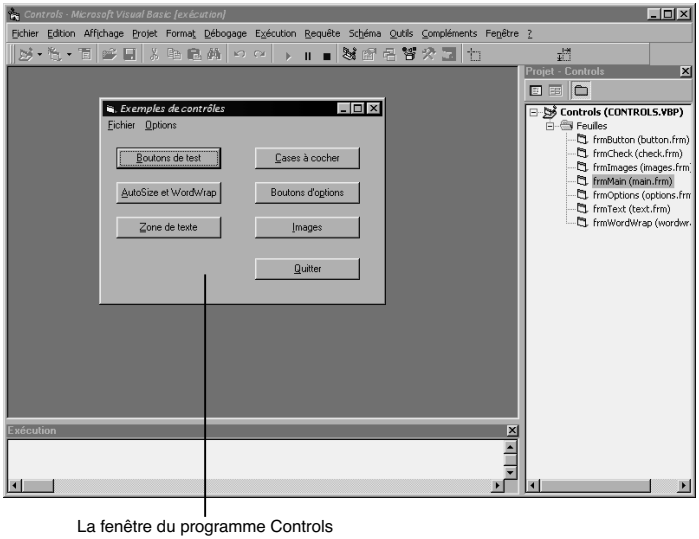
1. Lancez Visual Basic.
2. Insérez dans le lecteur le CD-ROM 1 de MSDN.
3. Cliquez sur l'onglet Existant et accédez au dossier `Samples\Vb98\Controls` via la boîte de dialogue Ouvrir un projet.
4. Double-cliquez sur l'icône Controls pour ouvrir le projet nommé Controls. Selon vos paramètres d'affichage Windows, l'extension `.VBP` peut ne pas apparaître dans la boîte de dialogue Ouvrir un projet. Dans tous les cas, que vous voyiez ou non l'extension dans les boîtes de dialogue, vous pouvez distinguer les types de fichiers d'après l'icône qui se trouve à gauche de leur nom. La boîte de dialogue Ouvrir un projet n'affiche que les fichiers de projets (à moins que vous ne modifiez la sélection dans la liste déroulante Type de fichier). L'icône que vous voyez vis-à-vis du projet Controls est l'icône qui représente tous les fichiers de projets Visual Basic.
5. Une fois le projet Controls ouvert, une boîte de dialogue s'affiche et demande si vous souhaitez ajouter au projet quelque chose du nom de *SourceSafe*. Pour tous les exercices de ce livre, vous répondrez non à cette question. Deux autres boîtes de dialogue reliées à SourceSafe apparaissent ensuite ; cliquez sur OK pour les fermer.

SourceSafe est un outil de Visual Studio (disponible pour tous les langages de la suite) qui permet de garder la trace des différentes versions de vos programmes sources.

Le programme source est l'ensemble du code et des éléments visuels qui constituent l'application que vous écrivez. Ce programme source n'est pas distribué, car il n'est que votre modèle de travail. Seule l'application finalisée et compilée doit être distribuée.

6. L'ouverture du projet Controls ne change pas grand-chose à l'écran Visual Basic. L'application Controls ne s'exécute pas, car vous n'avez fait que charger le projet depuis le disque. Pour lancer l'application (de façon interprétée), choisissez Exécution, Exécuter ; le résultat est montré en Figure 2.8.

Figure 2.8
Le programme Controls s'exécute dans l'environnement Visual Basic.



Lorsque vous exécutez une application interprétativement, la fenêtre du programme demeure dans l'environnement Visual Basic actif, de sorte que vous pouvez lui apporter des modifications ou corriger des erreurs en cours de route. Une fois compilé, le programme s'exécute directement dans Windows, hors de l'environnement Visual Basic.

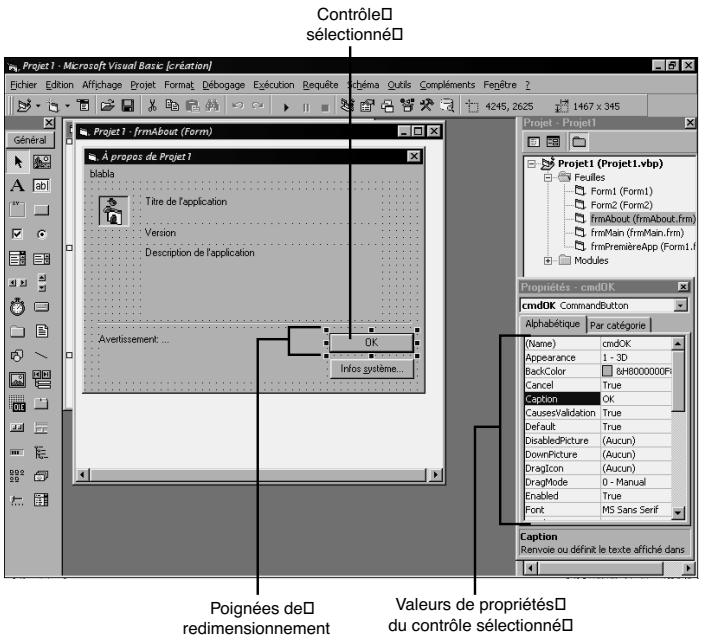
Plutôt que de passer par la commande menu Exécution, Exécuter, vous pouvez lancer le programme en appuyant sur F5 ou en cliquant sur le bouton de barre d'outils Exécuter.

7. Le programme Controls offre une démonstration de plusieurs des contôles disponibles dans la Boîte à outils Visual Basic. Cliquez sur un bouton, puis testez les options qui apparaissent.

8. Après avoir essayé les diverses options du programme, cliquez sur le bouton Arrêt pour interrompre l'exécution du programme et pour fermer la fenêtre. Vous voilà de retour dans l'environnement Visual Basic. A ce point, le programme Controls est toujours chargé dans l'environnement, mais l'application elle-même n'est pas active. Vous êtes maintenant libre d'étudier les diverses fenêtres Visual Basic.
9. Jetez un œil à la fenêtre Projet. Vous constatez que le programme Controls est constitué de feuilles uniquement. Le code existe bel et bien (cliquez sur le bouton Code pour afficher la fenêtre Projet ; cliquez de nouveau sur la fenêtre Afficher l'objet pour revenir à la liste des feuilles), mais il réside dans chacun des sept fichiers de feuilles qui accompagnent le projet.
10. Dans la fenêtre Projet, cliquez sur le nom d'une feuille pour la faire apparaître dans la zone d'édition de la fenêtre Feuilles. La feuille est semblable à ce qui apparaissait lors de l'exécution du programme. Consultez la fenêtre Présentation des feuilles (si vous ne la voyez pas, sélectionnez Affichage, Fenêtre Présentation des feuilles) pour connaître la position de la feuille sur l'écran tel qu'il apparaîtra lors de l'exécution.
11. Faites glisser l'icône miniature de la feuille depuis la fenêtre Présentation des feuilles jusqu'à un emplacement différent. Si vous lancez le programme, la fenêtre de la feuille en question apparaîtra là où vous l'avez fait glisser.
12. Examinez la fenêtre Propriétés, qui affiche les valeurs de propriétés du contrôle sélectionné. Gardez à l'esprit que le contenu de la fenêtre Propriétés ne concerne que le contrôle individuel sélectionné dans la fenêtre Feuilles. Sur la Figure 2.9, la fenêtre Propriétés affiche les propriétés du bouton d'option sélectionné (entouré de huit poignées de redimensionnement).
13. Faites défiler les valeurs de propriétés pour visualiser toutes les propriétés du contrôle sélectionné.
14. Cliquez sur un autre contrôle de la feuille et examinez la fenêtre Propriétés mise à jour. Lorsque vous sélectionnez un contrôle en cliquant dessus, des poignées de redimensionnement apparaissent tout autour, et la fenêtre Propriétés est mise à jour pour refléter les propriétés dudit contrôle.

Laissez le projet ouvert pour la prochaine section. En revanche, vous pouvez fermer les barres d'outils Edition et Editeur de code de feuille (si elles sont affichées), car nous n'en aurons pas besoin. Maintenant que vous avez fait connaissance de l'environnement Visual Basic, vous êtes prêt à y échafauder vos propres créations.

Figure 2.9
La fenêtre Propriétés affiche les valeurs de propriétés du contrôle sélectionné.



Créer une application à partir de zéro

Cette section conclut le chapitre en vous guidant dans la création d'une application. Vous ne comprendrez peut-être pas tout ce qui va se passer ; mais vous allez tout de même au bout de l'exercice, vous n'en serez que mieux paré pour la suite. Cette première application est simple, mais elle illustre bien la facilité avec laquelle on crée les programmes Visual Basic. C'est dans la suite de cet apprentissage que vous approfondirez les détails de ce que nous ne faisons que présenter aujourd'hui.

Configurer la feuille

Notre première application se contentera d'afficher une image et un bouton de commande. Le programme modifie l'image lorsqu'on clique sur le bouton. Pour créer cette simplissime application, suivez ces étapes :

1. Sélectionnez Fichier, Nouveau projet pour afficher la boîte de dialogue Nouveau projet. Si une application est déjà ouverte, Visual Basic demande si vous souhaitez enregistrer les modifications. Cliquez sur Non.

2. Cliquez sur l'icône EXE standard. L'environnement Visual Basic ne contient, alors, qu'une seule feuille nommée `Form1` (ce qu'indique la barre de titre). La feuille s'affiche sur l'arrière-plan de la zone d'édition, blanche, de la fenêtre Feuilles.

3. Cliquez sur le bouton Agrandir pour donner à la zone d'édition de la fenêtre Feuilles (c'est-à-dire l'arrière-plan blanc, et non la feuille grise elle-même) sa taille maximale. Ce qui libère assez d'espace pour agrandir la feuille.

Des poignées de redimensionnement apparaissent autour de la feuille parce que la feuille est le seul objet présent dans la zone d'édition de la fenêtre Feuilles. Vous pouvez remarquer que la fenêtre Propriétés affiche les propriétés de la feuille. Comme tout objet, chaque feuille a des valeurs de propriétés paramétrables.

4. Faites glisser vers le coin inférieur droit de l'écran la poignée de redimensionnement située en bas à droite de la feuille. Au fur et à mesure, vous pouvez voir, à droite de la barre d'outils, les dimensions de la feuille se modifier. Agrandissez la feuille jusqu'à une taille d'environ 7400 *twips* sur 5200. Cette opération définit l'arrière-plan du programme ; la Figure 2.10 montre le résultat. (La fenêtre Présentation des feuilles peut apparaître sous la fenêtre Propriétés.)

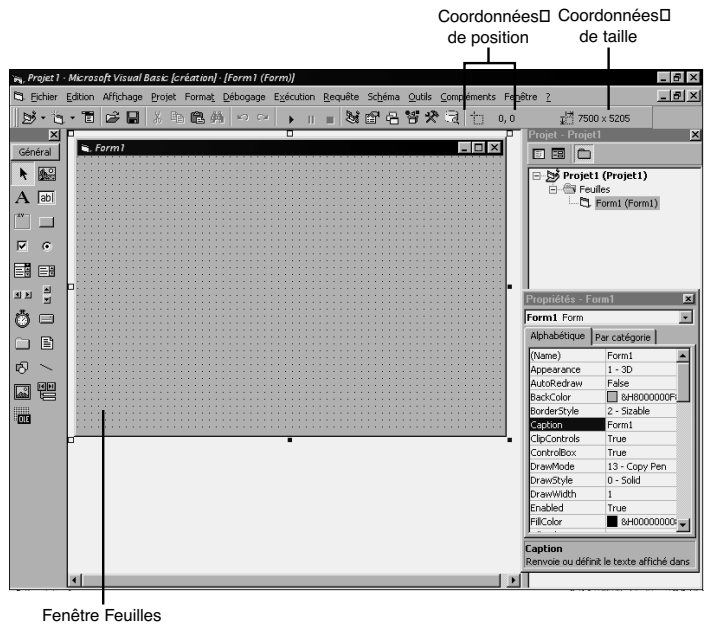


Figure 2.10
En redimensionnant la fenêtre Feuilles, vous redimensionnez la fenêtre d'application de votre programme.

Le twip est une mesure d'affichage. On peut se représenter le twip comme un point de l'écran ; mais différents moniteurs et cartes vidéo donnent différentes résolutions, donc un nombre différent de points. Le twip est une unité qui mesure, de façon indépendante de la résolution réelle, un point imaginaire de l'écran (point plus petit que ne le permettent les résolutions les plus hautes). En conséquence, une feuille mesurant 7400 twips n'occupera pas 7400 points réels de l'écran (ou pixels).

En positionnant et en dimensionnant les fenêtres Feuilles, vérifiez les coordonnées correspondantes à droite de la barre d'outils. Ces valeurs sont affichées par paires. La première valeur des coordonnées de position équivaut au nombre de twips entre le bord gauche de l'écran et le côté de la fenêtre. La seconde valeur équivaut au nombre de twips entre le bord supérieur de l'écran et le haut de la fenêtre. La seconde paire de valeurs, les coordonnées de taille, équivalent au nombre de twips que la fenêtre occupe, en largeur et en hauteur respectivement. Les propriétés correspondant aux coordonnées de position sont nommées `Left` *et* `Top`*. Les propriétés correspondant aux coordonnées de taille sont nommées* `Width` *et* `Height`*. Visual Basic met automatiquement à jour ces valeurs dans la fenêtre Propriétés lorsque vous déplacez ou redimensionnez la feuille dans la zone d'édition de la fenêtre Feuilles.*

5. Sélectionnez Affichage, Fenêtre Présentation des feuilles pour afficher la fenêtre. Dans la fenêtre Présentation des feuilles, centrez l'écran miniature de sorte que la fenêtre d'application se place au milieu de l'écran lorsque le programme démarre. Bien que la fenêtre Présentation des feuilles elle-même ne change pas, les coordonnées de position refléteront la modification.
6. Fermez la fenêtre Présentation des feuilles afin de laisser de la place aux autres fenêtres.

Les points qui constellent la feuille forment la grille. L'affichage de cette grille peut être activé ou désactivé : choisissez Outils, Options, cliquez sur l'onglet Général, puis cochez ou décochez la case Afficher la grille. La grille n'apparaîtra pas lors de l'exécution du programme ; elle n'est là que pour vous aider à placer et à dimensionner les contrôles sur la feuille.

La grille est la trame de points qui constitue l'arrière-plan de la fenêtre Feuilles. La densité de points de la grille peut être paramétrée par la commande Outils, Options.

7. Attribuez à la feuille un nom plus parlant que `Form1`. Cette opération vous permettra au passage de travailler dans la fenêtre Propriétés. Le nom de la feuille sélectionnée est défini dans la propriété (`Name`), qui est mis entre parenthèses afin de maintenir le nom au sommet de la liste alphabétique des propriétés. (A partir de maintenant, nous ferons abstraction de ces parenthèses.) Si la propriété `Name` n'est pas encore visible, faites défiler la fenêtre Propriétés ; notez que la valeur de `Name` est actuellement `Form1`.
8. Sélectionnez la propriété `Name` de la feuille et tapez `frmMyFirst`. Le nouveau nom s'affiche aussitôt à droite de la propriété `Name`, ainsi que sur la barre de titre de Visual Basic.

Vous modifierez et attribuerez les différentes valeurs de la fenêtre Propriétés de la même manière que vous venez de changer le nom de la feuille. Faites défiler la fenêtre jusqu'à la propriété désirée, sélectionnez-la, puis saisissez la nouvelle valeur (ou bien choisissez parmi les options proposées dans les listes déroulantes).

9. Modifiez la barre de titre de la feuille : sélectionnez la propriété `Caption` et tapez **Bonne Journée**. La propriété `Caption` définit ce qui s'affiche dans la barre de titre de la feuille lorsque l'utilisateur lance le programme. Le nouveau nom apparaît à la fois dans la fenêtre Propriétés et sur la barre de titre de la feuille.
10. Avant d'aller plus loin, il est prudent d'enregistrer la feuille sur le disque. Choisissez Fichier, Enregistrer le projet. Cette commande enregistre tous les fichiers inclus dans le projet en cours (lequel ne contient pour l'instant qu'une seule feuille), ainsi que les fichiers de description du projet sous extension `.VBP`. Visual Basic demande d'abord le nom de fichier qui doit être attribué à la feuille. La valeur de la propriété `Name` de la feuille sert de nom par défaut. Si vous acceptez ce nom par défaut, ce que nous vous invitons à faire, Visual Basic y ajoute l'extension `.FRM`. (Si votre projet contenait plusieurs feuilles, modules ou autres types d'objets stockés dans des fichiers, vous auriez eu à spécifier un nom pour chacun.) Visual Basic s'enquiert ensuite du nom de projet pour le fichier de description. Nommez le projet `HappyApp`, puis enregistrez. S'il vous est proposé d'ajouter le projet à la bibliothèque SourceSafe, répondez Non.

Le fichier de description de projet est ce que vous chargerez pour travailler sur l'application par la suite. Lorsque vous ouvrez ce fichier de description, Visual Basic charge tous les fichiers associés au projet, et en affiche les noms dans la fenêtre Projet.

Ajouter les détails

Maintenant que l'arrière-plan de l'application est créé, il vous reste à ajouter les détails, c'est-à-dire à disposer les contrôles sur la feuille. Ce qui se fait généralement de la manière suivante :

1. Sélectionnez le contrôle dans la Boîte à outils.
2. Placez le contrôle à la position voulue.
3. Dimensionnez le contrôle.
4. Définissez les valeurs de propriétés du contrôle.
5. Activez le contrôle à l'aide de code Visual Basic, si nécessaire.

Les étapes suivantes vous enseignent à sélectionner les contrôles dans la Boîte à outils et à les disposer sur la feuille. Dans la plupart des cas, vous procéderez de l'une de ces deux manières :

- Double-cliquez sur l'icône du contrôle dans la Boîte à outils. Visual Basic place alors ce contrôle au centre de la feuille. Vous pouvez ensuite l'affecter à la position choisie, et le redimensionner en faisant glisser les poignées de redimensionnement.
- Cliquez sur l'icône du contrôle dans la Boîte à outils, puis déplacez vers l'emplacement voulu le curseur cruciforme qui apparaît. Là, cliquez, puis maintenez le bouton en déplaçant la souris ; vous dimensionnez le contrôle. Lorsque le contrôle a la position et la taille souhaitées, lâchez le bouton de la souris.

Les étapes qui suivent enrichissent l'application créée à la section précédente :

1. Double-cliquez sur le contrôle Label pour placer un label au centre de votre feuille. Sur le contrôle Label figure la lettre A, comme nous l'avons vu à la section sur la barre d'outils. (Rappelez-vous que les info-bulles vous indique la fonction des icônes que vous ne reconnaissez pas.) Le label est maintenant l'outil sélectionné dans la zone d'édition de la fenêtre Feuilles, et des poignées de redimensionnement apparaissent tout autour. La fenêtre Propriétés, quant à elle, s'actualise pour afficher les propriétés du label ; sur la barre d'outils, les coordonnées de position et de taille reflètent les nouvelles mesures du contrôle.

   On se sert d'un label pour afficher du texte sur une feuille. Dans ce cas précis, le label constituera une zone de titre pour l'application.

2. Faites glisser le label sur la feuille jusqu'à ce qu'il se situe environ à 1320 twips du bord gauche de la fenêtre Feuilles et 120 twips du bord supérieur. Guidez-vous à l'aide des coordonnées affichées sur la barre d'outils.

A moins que vous ne modifiez l'option Aligner les contrôles sur la grille (onglet Général de la boîte de dialogue Outils, Options), Visual Basic aligne automatiquement les contrôles sur les points les plus proches, afin que l'ensemble soit proprement ordonné.

3. Double-cliquez sur le contrôle Bouton de commande de la Boîte à outils, afin de placer un bouton de commande au centre de la feuille.
4. Cherchez sur la Boîte à outils le contrôle Image, puis faites un simple clic sur son icône (un double clic centrerait automatiquement le contrôle). Placez le curseur sur la feuille et dessinez le contrôle Image, en essayant de l'ancrer environ à 2520 twips du bord gauche et 2880 twips du bord supérieur. Donnez à l'image une taille approximative de 2175 twips en largeur et 1825 twips en hauteur. Si vous ne déplacez pas les poignées trop rapidement, une info-bulle s'affiche pour vous indiquer les coordonnées de l'image. Lorsque l'objet a atteint la taille voulue, relâchez le bouton de la souris. La Figure 2.11 montre l'écran à cette étape. Lorsque vous exécuterez le programme, le contrôle affichera une image.

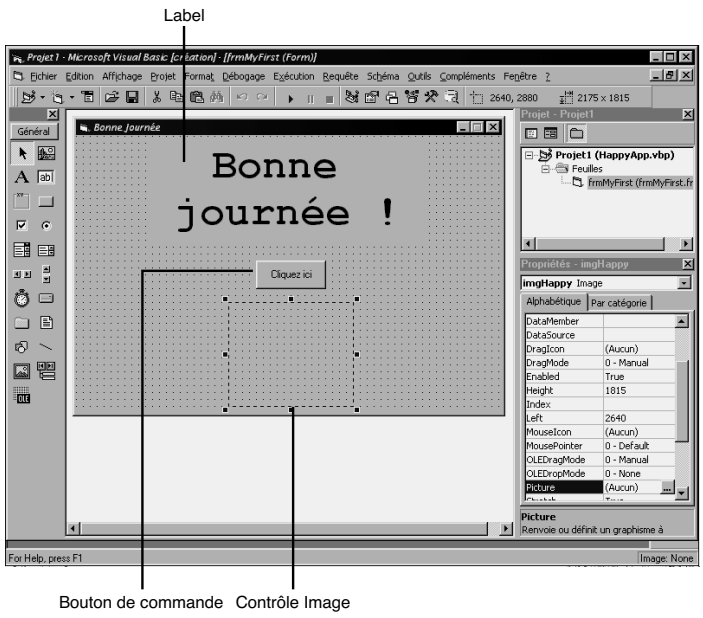


Figure 2.11
Votre application prend forme.

En suivant les étapes qui précèdent, vous placez le contrôle Image approximativement à la position demandée. Pour lui affecter les coordonnées de position et de taille exactes, il suffit de donner les valeurs spécifiées aux propriétés correspondantes : `Left = 2520, Top = 2880, Width = 2175` *et* `Height = 1825`*. A l'avenir, c'est par une notation de ce type que nous vous indiquerons les valeurs de propriétés. Vous savez maintenant que, pour assigner de nouvelles valeurs, il suffit de cliquer sur le nom de la propriété concernée et de saisir directement.*

Les coordonnées de position et de taille sont toujours spécifiées en twips, et par paires. Vous les verrez souvent énoncées entre parenthèses, à la manière de points cartésiens : `(2520, 2880)`*.*

5. Même si vous ne comprenez pas encore chacune des propriétés, vous êtes en mesure de leur attribuer des valeurs. Il s'agit, à présent, de définir de nouvelles valeurs de propriétés pour la feuille et ses contrôles afin de finaliser l'aspect de l'application. Lorsque cela sera fait, il restera à ajouter le code propre à connecter les divers composants et à les faire fonctionner ensemble.

   Le Tableau 2.2 présente une liste des valeurs de propriétés qu'il faut maintenant définir pour les trois contrôles et la feuille elle-même. Rappelez-vous qu'il faut sélectionner la feuille ou le contrôle spécifique avant de pouvoir en modifier les valeurs de propriétés. Pour sélectionner une feuille, cliquez n'importe où à l'intérieur ou sur la barre de titre — , mais pas sur l'un des contrôles. La fenêtre Propriétés s'actualise pour renvoyer les nouvelles valeurs. Cliquez d'abord sur le label, le bouton de commande ou l'image pour sélectionner un contrôle, et modifiez-le en sélectionnant une propriété, puis en saisissant la nouvelle valeur.

Au premier abord, le paramétrage des informations de police pour un contrôle peut sembler confus. Lorsque vous sélectionnez la propriété `Font` *d'un contrôle, des points de suspension apparaissent après la valeur. Ces points de suspension indiquent que vous pouvez attribuer à cette propriété plus d'une valeur ; en cliquant sur les points, vous affichez la boîte de dialogue reproduite en Figure 2.12. Une fois que vous avez défini les valeurs dans cette boîte de dialogue et validé par OK, les valeurs de diverses propriétés liées à la police se modifient en conséquence.*

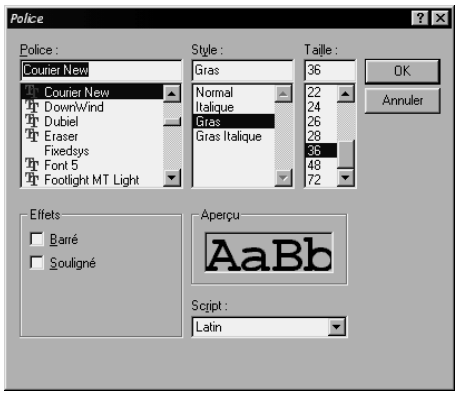


Figure 2.12
La boîte de dialogue Police vous permet de définir plusieurs valeurs pour la propriété Font.

Tableau 2.2 : Affectez les valeurs de propriétés suivantes à la feuille et aux contrôles de l'application

Contrôles	*Propriétés*	*Valeurs de propriétés*
Feuille	`Max Button`	`False` (ouvrez la liste déroulante pour afficher les valeurs)
Label	`Alignment`	`Center` (ouvrez la liste déroulante pour afficher les valeurs)
Label	`Name`	`lblHappy`
Label	`Caption`	`Bonne journée!`
Label	`Font`	`Courier New`
Label	`Font style`	`Bold`
Label	`Size`	`36`
Label	`Left`	`1320`
Label	`Height`	`1695`
Label	`Top`	`120`

Tableau 2.2 : Affectez les valeurs de propriétés suivantes à la feuille et aux contrôles de l'application *(suite)*

Contrôles	*Propriétés*	*Valeurs de propriétés*
Label	`Width`	`4695`
Image	`Name`	`imgHappy`
Image	`Stretch`	`True`
Bouton de commande	`Name`	`cmdHappy`
Bouton de commande	`Caption`	`Cliquez ici`

Vous pouvez, dans le cours même de l'écriture, exécuter l'application pour voir un peu ce que ça donne. Si, par exemple, vous appuyez maintenant sur F5 (raccourci pour la commande Exécuter), Visual Basic analyse le programme et affiche une fenêtre d'application active incluant un bouton de commande sur lequel vous pouvez cliquer. Rien ne se passe lorsque vous cliquez sur le bouton, sinon que le bouton s'anime pour figurer le clic. Par ailleurs, le contrôle Image que nous avons placé ne contient pour l'instant rien du tout. Nous réglerons ces deux problèmes mineurs dans la section suivante. Pour quitter le programme, cliquez sur le bouton de fermeture de la fenêtre d'application. La prochaine leçon vous apprendra à ajouter des "portes de sorties" plus commodes.

Finaliser par le code

Grâce aux instructions Visual Basic que nous allons maintenant ajouter, votre application va devenir tout à fait fonctionnelle, quoique simple. Cette procédure peut vous sembler étrange, car il s'agit de taper des codes quelque peu ésotériques dans une fenêtre Code qui s'affiche de façon impromptue. Observez les quelques étapes qui suivent ; les prochains chapitres vous apprendront plus de subtilités sur la question.

1. Double-cliquez sur la forme, soit quelque part sur la grille dans la fenêtre Feuilles. La feuille disparaît et la fenêtre Code s'affiche avec ces deux lignes :

```
Private Sub Form_Load()
End Sub
```

Ces lignes font partie des quatre lignes indispensables au code d'une feuille. La fenêtre Feuilles fonctionne comme un petit traitement de texte dans lequel vous pouvez insérer, supprimer et modifier les différentes instructions contenues dans le programme.

Le code apparaît sous forme de procédures ; chaque procédure nécessite une ligne de début et une ligne de fin, qui en désignent les points de départ et d'arrêt. Pour la plupart des procédures, Visual Basic ajoute automatiquement ces deux lignes.

Une procédure est une section de code Visual Basic qui contient des instructions chargées d'une tâche précise ; par exemple, centrer une fenêtre Feuilles.

2. Vous allez maintenant saisir vos premières lignes de code. Insérez trois espaces au début de chaque ligne, de sorte qu'elles soient légèrement décalées par rapport aux lignes de début et de fin. Cet ensemble de renfoncements, que les programmeurs nomment *indentation*, permettent de circonscrire chaque procédure à l'intérieur d'une longue succession. Le code qui suit contraint la fenêtre d'application à s'afficher au centre de l'écran, quelle que soit la résolution utilisée.

```
frmMyFirst.Left = (Screen.Width - frmMyFirst.Width) / 2
frmMyFirst.Top = (Screen.Width - frmMyFirst.Height) / 2
```

   A peine avez-vous commencé de taper la première ligne que Visual Basic affiche un *complément automatique d'instructions* (voir Figure 2.13). Lorsque Visual Basic "pressent" que vous allez spécifier des valeurs de propriétés, il propose sous forme de liste déroulante les différentes options disponibles pour le contrôle ; vous pouvez ainsi choisir dans la liste plutôt que de saisir le nom de la propriété en entier. Dès que vous avez sélectionné une propriété et validé en appuyant sur la barre d'espace, Visual Basic insère le nom complet, et vous pouvez poursuivre votre saisie.

3. Dans la fenêtre Projet, cliquez sur le bouton Afficher l'objet afin de revenir à la fenêtre Feuilles.

4. Double-cliquez sur le bouton de commande pour ouvrir la fenêtre Code de nouveau. Au code initial vient s'ajouter un nouvel ensemble d'instructions de début et de fin ; il s'agit d'une nouvelle procédure destinée à gérer le bouton de commande. Entre ces deux nouvelles lignes, tapez le code suivant, toujours précédé de trois espaces :

```
imgHappy = LoadPicture("\Program Files\Microsoft Visual Studio\
➥Common\Graphics \Bitmaps\Assorted\Happy.bmp")
```

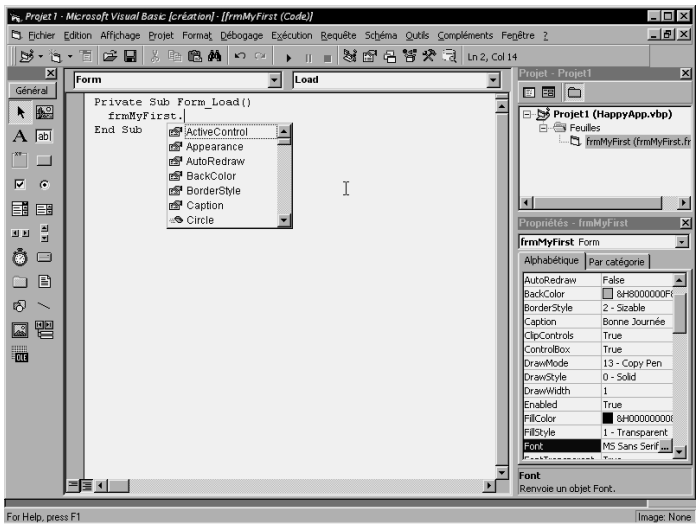


Figure 2.13
Visual Basic accélère la saisie du code.

Alors que vous tapez la parenthèse ouvrante après `LoadPicture`, Visual Basic propose une aide contextuelle semblable au Complément automatique d'instructions que nous avons déjà vu. Certaines instructions Visual Basic, notamment celles que l'on écrit entre parenthèses, requièrent que vous saisissiez une ou plusieurs valeurs. Visual Basic proposant automatiquement le format de ces valeurs, vous saurez toujours combien il en faut. A mesure que vous progresserez dans votre connaissance du langage, vous comprendrez mieux la nécessité de ces valeurs. Dans un langage aussi complet que Visual Basic, l'aide contextuelle proposée s'avère d'une grande utilité.

5. Exécutez le programme et cliquez sur le bouton de commande : une image apparaît (voir Figure 2.14). Bravo ! Vous avez réussi à créer une application complète sans recourir à l'assistant Création d'applications. Cette application affiche une image lorsqu'on clique sur un bouton ; elle contient du code, et vous en avez paramétré chacun des contrôles.
6. Cliquez sur le bouton de fermeture pour quitter le programme. Assurez-vous de bien sauvegarder votre projet avant de quitter Visual Basic.

Figure 2.14
Votre application affiche une image lorsqu'on clique sur le bouton.

En résumé

Si, dans ce chapitre, nous nous sommes attaché à décrire l'environnement Visual Basic, c'est que vous ne deviendrez un programmeur Visual Basic efficace qu'après avoir compris le jeu des diverses fenêtres et interfaces. Visual Basic propose plusieurs niveaux d'aide, incluant l'aide en ligne, le support Web et un support technique personnalisé disponible en plusieurs formules. Vous avez ainsi une assistance précieuse à portée de main, au cas où un aspect de l'interface ou du langage Visual Basic vous poserait un problème.

Pour créer une application, il faut créer un nouveau projet, ajouter des contrôles dans la fenêtre Feuilles, définir les valeurs de propriétés de la feuille et des contrôles, et activer ces contrôles à l'aide du code. Le projet d'aujourd'hui était exceptionnellement simple, en raison notamment du peu de lignes de code requises : trois.

La leçon de demain répondra à quelques questions que vous vous posez à propos des contrôles et de leurs propriétés.

Questions-réponses

Q Quelles doivent être les dimensions de ma fenêtre Feuilles ?

R C'est la finalité de l'application qui doit décider de la grandeur de la fenêtre Feuilles et du nombre de feuilles requis. Pour des programmes simples, une feuille est généralement suffisante ; la taille de la feuille, quant à elle, dépend du nombre de contrôles qu'elle doit recevoir, et de la nature du programme.

Nous aurions pu, dans ce chapitre, créer une feuille de taille maximale ; mais, avec seulement trois contrôles, c'eût été tout à fait démesuré.

Q A quoi exactement sert le code que nous avons écrit ?

R Le code est nécessaire au bon fonctionnement de l'application. La ligne contenant le mot clé `LoadPicture` est indispensable, car c'est elle qui lance le chargement de l'image lorsqu'on clique sur le bouton de commande. Les deux autres lignes, que nous avons ajoutées après avoir double-cliqué sur la fenêtre Feuilles pour ouvrir la fenêtre Code, servent à centrer la forme quelles que soient la taille et la résolution de l'écran utilisé.

Q Si c'est le code qui centre la fenêtre Feuilles, était-il nécessaire d'utiliser la fenêtre Présentation des feuilles ?

R Quelque position que vous donniez à la feuille dans la fenêtre Présentation des feuilles, les deux lignes de la procédure centrent automatiquement la feuille au lancement du programme.

La fenêtre Présentation des feuilles est une ébauche qui vous donne une idée de l'emplacement de la fenêtre Feuilles lorsque la feuille est chargée. Pour un contrôle plus serré, et notamment si vous n'êtes pas certain de la taille de l'écran sur lequel sera exécuté le programme, la feuille doit être positionnée à l'aide du code.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices grâce auxquels vous mettrez en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quelle est la différence entre la Boîte à outils et la barre d'outils ?
2. Quel est le nom du service d'aide fourni sur abonnement aux programmeurs Microsoft ?
3. La fenêtre Feuilles ne peut contenir qu'une feuille à la fois. Vrai ou faux ?
4. Que se passe-t-il lorsqu'on clique sur un contrôle de la Boîte à outils ?
5. Que se passe-t-il lorsqu'on double-clique sur un contrôle de la Boîte à outils ?
6. C'est depuis la fenêtre Boîte à outils que l'on définit les propriétés d'un contrôle. Vrai ou faux ?
7. Comment Visual Basic détermine-t-il de quel contrôle les propriétés doivent être affichées dans la fenêtre Propriétés ?
8. Que signifient les points de suspension dans une valeur de la fenêtre Propriétés ?
9. Quel est le nom de la propriété qui spécifie le titre d'un bouton de commande ?
10. Pourquoi faut-il changer le nom par défaut des contrôles ?

Exercice

Chargez l'application créée aujourd'hui pour pouvoir la modifier. Ajoutez un peu de couleur en appliquant du bleu à l'arrière-plan de la feuille. Placez également sur la feuille un nouveau bouton de commande, labelisé `Quitter`, et spécifiez une propriété `Name` adéquate pour ce contrôle. Dans la nouvelle procédure qui s'affiche dans la fenêtre Code, ajoutez la ligne suivante, qui gère le nouveau bouton :

```
End
```

Exécutez l'application, puis cliquez sur le bouton Quitter. N'est-ce pas une façon plus élégante de prendre congé d'une application ?

Chapitre 3

Gestion des contrôles

Maintenant que vous avez créé deux applications, l'une avec l'assistant Création d'applications et l'autre à la force de votre poignet (façon de parler), il est temps d'étudier le fonctionnement interne de Visual Basic. Les travaux pratiques des deux premiers chapitres ont démontré la facilité avec laquelle on peut créer des programmes, et ils vous ont donné un aperçu de l'environnement Visual Basic. A partir d'aujourd'hui, vous allez commencer à comprendre de quelle façon les composants fonctionnent ensemble — notamment les contrôles et leurs propriétés.

Voici ce que nous découvrirons aujourd'hui :

- les propriétés des contrôles courants ;
- pourquoi les contrôles ont autant de propriétés ;
- les outils communs de la Boîte à outils ;
- comment les touches de raccourcis accélèrent la saisie des données ;
- comment le focus aide l'utilisateur à sélectionner les contrôles ;
- le fonctionnement des procédures.

Etude des contrôles

La Boîte à outils contient un ensemble de contrôles dont vous vous servez dans vos applications. Dans la Boîte à outils, les contrôles sont en nombre illimité : vous pouvez double-cliquer sur le contrôle Image autant de fois que votre application le nécessite ; chaque fois, un nouveau contrôle est créé sur la feuille.

La variété des contrôles (ou outils) disponibles dans la Boîte à outils dépend des exigences de votre application. Dans le cadre de votre apprentissage, la Boîte à outils contiendra, en général, le même jeu d'outils que nous avons vu dans la leçon d'hier. Il s'agit des contrôles standards que Visual Basic charge lorsque vous créez une nouvelle application.

Un contrôle n'est pas accessible à votre application tant qu'il n'apparaît pas dans la Boîte à outils. Vous ne pouvez donc pas, par exemple, ajouter sur votre feuille un contrôle de navigation Internet si ce contrôle n'est pas proposé par la Boîte à outils. Si vous utilisez l'assistant Création d'applications, il ajoutera à la Boîte à outils tous les contrôles requis pour générer le projet.

Si, dans une des leçons de ce livre, il est fait référence à un contrôle absent de la Boîte à outils standard, nous vous expliquerons comment vous le procurer.

Comme le montre la Figure 3.1, la Boîte à outils peut devenir passablement encombrée à partir d'un certain nombre de contrôles. Une telle Boîte à outils accapare beaucoup d'espace dans votre fenêtre. Pour libérer de l'espace à gauche de votre écran, vous pouvez déplacer la Boîte à outils en en faisant glisser la barre de titre vers une nouvelle position. La fenêtre Boîte à outils peut également être redimensionnée.

Ne pas faire
Evitez de donner à la Boîte à outils une taille si réduite qu'elle en devienne invisible. Si la Boîte à outils est si petite que des contrôles ne puissent plus être vus, aucune barre de défilement ne s'affichera. Pour utiliser les outils situés au-delà des limites de la fenêtre, il faut étendre la fenêtre jusqu'à ce qu'ils redeviennent accessibles.

Les puristes Visual Basic appliquent aux jeux de contrôles disponibles une terminologie plus stricte. Les contrôles qui apparaissent en premier dans la Boîte à outils sont nommés contrôles intrinsèques. Les contrôles ActiveX, d'extension `.OCX`*, sont des contrôles externes qui peuvent être ajoutés à la Boîte à outils. Les contrôles insérables sont des contrôles créés dans des applications externes, telles que Microsoft Excel.*

Figure 3.1
La fenêtre Boîte à outils doit être d'une taille raisonnable.

Les propriétés sont les mêmes pour la plupart des contrôles. Le Tableau 3.1 liste les propriétés les plus courantes. Vous comprendrez sans doute pourquoi ces propriétés s'appliquent à tant de contrôles. Tous les contrôles doivent avoir une position sur l'écran (définie par les propriétés `Left` et `Top`), et la plupart ont des couleurs de premier et d'arrière-plan, ainsi que des propriétés de police, pour ceux qui incluent du texte.

Tableau 3.1 : Propriétés communes à plusieurs contrôles Visual Basic

Propriétés	*Description*
`Alignment`	Détermine si le texte du contrôle, tel qu'un label ou un bouton de commande, apparaît comme aligné à gauche, centré ou aligné à droite.
`BackColor`	Spécifie la couleur de l'arrière-plan du contrôle. Vous choisissez cette couleur dans une palette en ouvrant la liste déroulante de la propriété.
`BorderStyle`	Détermine si le contrôle est entouré d'une bordure.
`Caption`	Contient le texte affiché sur le contrôle.
`Enabled`	Définie *via* une liste déroulante, cette propriété sera `True` (vraie) si le contrôle doit être accessible à l'utilisateur, et `False` (fausse) si le contrôle ne doit pas être accessible. Cette propriété permet d'activer et de désactiver les contrôles selon qu'on les veut ou non disponibles lors de l'exécution du programme.

Tableau 3.1 : Propriétés communes à plusieurs contrôles Visual Basic *(suite)*

Propriétés	*Description*
`Font`	Affiche une boîte de dialogue dans laquelle vous paramétrez pour le texte d'un contrôle divers attributs de police, tels que la taille ou le style.
`ForeColor`	Spécifie la couleur de premier plan du contrôle. Vous choisissez cette couleur dans une palette en ouvrant la liste déroulante de la propriété.
`Height`	Spécifie la hauteur en twips du contrôle.
`Left`	Définit, à partir du bord gauche de la feuille, l'origine horizontale du contrôle. Pour une feuille, la propriété `Left` définit la distance (en twips) au bord gauche de l'écran.
`MousePointer`	Détermine la forme que prend le curseur lorsque l'utilisateur déplace la souris sur le contrôle.
`Name`	Spécifie le nom du contrôle. Come nous l'avons vu dans la leçon d'hier, la fenêtre Propriétés affiche la propriété `Name` entre parenthèses afin qu'elle apparaisse en premier dans la liste.
`ToolTipText`	Contient le texte qui apparaît lorsque l'utilisateur maintient le curseur sur un contrôle (info-bulle).
`Top`	Définit, à partir du bord supérieur de la feuille, l'origine verticale du contrôle. Pour une feuille, la propriété `Left` définit la distance (en twips) au bord supérieur de l'écran.
`Visible`	Définie *via* une liste déroulante, cette propriété sera `True` (vraie) si le contrôle doit être visible sur la feuille, et `False` (fausse) si le contrôle doit être masqué.
`Width`	Spécifie la largeur en twips du contrôle.

Rappelez-vous que les valeurs de propriétés sont définies pour chaque contrôle avant même que vous ne les modifiiez. Dès que vous placez un contrôle, Visual Basic lui applique un jeu de valeurs de propriétés prédéterminé (les valeurs les plus courantes). Visual Basic affecte également aux contrôles des noms et libellés, par défaut, que vous souhaiterez sans doute modifier. Les valeurs par défaut fonctionnent très bien dans la plupart des cas. Vous n'aurez jamais à changer *toutes* les valeurs de propriétés par défaut d'un contrôle.

Comme nous l'avons vu, les propriétés des contrôles peuvent être paramétrées lors de l'écriture du programme. Mais vous pouvez également définir et modifier les valeurs de propriétés dans le cours même de l'exécution du programme. Par exemple, la

propriété `Enabled` est souvent modifiée pendant l'exécution pour qu'un contrôle ne soit plus accessible à l'utilisateur. Vous pouvez ainsi désactiver un bouton de commande qui génère un rapport jusqu'à ce que l'utilisateur ait donné le contenu du rapport.

La fenêtre Propriétés n'affiche pas toutes les propriétés de tous les contrôles. N'y apparaissent pas les propriétés seulement définissables dans le code Visual Basic.

Certaines propriétés, telles que `Alignment`, peuvent sembler étrange, du fait que, dans leur liste déroulante, les valeurs sont précédées d'un chiffre. `Alignment`, par exemple, prend l'une de ces trois valeurs : `0-Left Justify`, `1-Right Justify` et `2-Center`. Vous pouvez sélectionner ces valeurs à l'aide de la souris sans tenir compte des chiffres ; vous pouvez également, après avoir ouvert la liste déroulante d'une propriété, taper le chiffre correspondant à la valeur voulue. Ces chiffes sont tout aussi pratiques lorsque vous définissez les propriétés d'un contrôle dans le code Visual Basic : il suffit d'attribuer la valeur numérique, au lieu de saisir la valeur entière (comme `0-Left Justify`). Ces chiffres constituent des raccourcis utiles lors de l'écriture du code, comme nous le verrons dans les leçons suivantes.

Les sections suivantes décrivent les contrôles standards les plus utiles, ainsi que plusieurs des propriétés importantes associées à ces contrôles. Nous ne vous présenterons pas aujourd'hui tous les contrôles de la Boîte à outils, mais seulement ceux que vous utiliserez lors de vos premières expériences en Visual Basic.

Nous ne donnerons pas non plus un compte rendu exhaustif de toutes les propriétés, parce que certaines de ces propriétés ne sont que très rarement utilisées. Bien des programmeurs Visual Basic ne connaissent même pas toutes les propriétés disponibles pour un contrôle. En général, si vous souhaitez donner à un contrôle un aspect précis, il existe une propriété qui vous permet de le faire. Le tour d'horizon des contrôles et de leurs propriétés que nous vous proposons maintenant vous aidera à mieux comprendre la fonction des contrôles, et les diverses propriétés qui leur sont associées.

Certains programmeurs préfèrent que la fenêtre Propriétés affiche les propriétés par catégories. Pour appliquer un tel classement, cliquez sur l'onglet Par catégories de la fenêtre Propriétés (voir Figure 3.2).

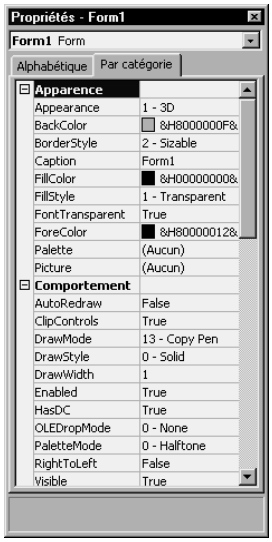


Figure 3.2
Vous pouvez classer les propriétés afin de les localiser plus rapidement.

Les propriétés de la feuille

Nombre des propriétés de feuille correspondent aux propriétés de contrôles présentées au Tableau 3.1. La feuille, en revanche, a cela de particulier qu'elle n'apparaît pas sur une feuille, mais directement sur l'écran de l'utilisateur. C'est pourquoi les propriétés `Left`, `Top`, `Width` et `Height` de la feuille sont fixées par rapport aux bords de l'écran, et non par rapport au bord d'une fenêtre Feuilles.

Les propriétés de feuille suivantes ont aussi leur importance :

- `BorderStyle`. Détermine la réaction de la fenêtre lorsque l'utilisateur cherche à la redimensionner. Les valeurs possibles incluent `0-None`, qui donne une fenêtre sans bordure ni barre de titre ; `1-Fixed Single`, qui donne une fenêtre non redimensionnable (l'utilisateur peut fermer la fenêtre, mais ni la redimensionner, ni la réduire, ni l'agrandir) ; `2-Sizable`, valeur par défaut qui donne une fenêtre redimensionnable classique, avec boutons Agrandir et Réduire.
- `ControlBox`. Les valeurs `True` et `False` de cette propriété déterminent l'affichage du *menu Système* de la feuille.

Le menu Système est le menu qui apparaît lorsqu'on clique sur l'icône située dans le coin supérieur gauche de la fenêtre. Ce menu propose les options classiques de déplacement, redimensionnement, réduction, agrandissement et fermeture.

- `Icon`. Spécifie un fichier d'icône pour le bouton qui représentera l'application sur la Barre des tâches Windows.
- `MaxButton`. Détermine l'affichage sur la fenêtre d'un bouton actif Agrandir.
- `MinButton`. Détermine l'affichage sur la fenêtre d'un bouton actif Réduire. (Si vous donnez la valeur `False` aux propriétés `MaxButton` et `MinButton`, aucun des deux boutons n'apparaîtra.)
- `Movable`. Détermine si l'utilisateur est libre de déplacer la feuille ou si cette feuille doit rester en place.
- `ShowInTaskbar`. Les valeurs `True` et `False` de cette propriété déterminent l'affichage de l'application sur la barre des tâches Windows.
- `StartUpPosition`. Offre un moyen rapide de définir la position de départ de la feuille sur l'écran. L'une des valeurs les plus utiles est `2-CenterScreen`, qui centre automatiquement la feuille dès sa première apparition.
- `WindowState`. Détermine la taille (normale, agrandie, réduite) de la feuille. Cette propriété est utile pour afficher par défaut la feuille réduite.

L'outil Pointeur

L'outil pointeur est le seul élément de la Boîte à outils qui ne soit pas un contrôle. Le pointeur, et c'est en fait sa seule utilité, permet de se débarrasser du curseur cruciforme, qui apparaît quand vous sélectionnez un contrôle de la Boîte à outils.

Le contrôle Label

Le contrôle Label permet d'afficher du texte. Votre utilisateur ne peut naturellement pas changer le texte d'un label. Mais vous pouvez, vous, modifier le label en cours d'exécution, dans le code. (Voir le Chapitre 5 pour plus de détails.) Les programmeurs ont souvent recours aux labels pour des titres, des invites ou des descriptions. Si, par exemple, vous souhaitez que l'utilisateur saisisse une valeur dans un autre contrôle, tel qu'une zone de texte, vous placez à côté de cette zone de saisie un label décrivant la valeur demandée. Sans le label, l'utilisateur ne saurait quelle valeur entrer.

`AutoSize` et `WordWrap` sont deux propriétés qui affectent l'affichage du texte sur un label. Si vous donnez à `AutoSize` la valeur `True`, la taille du label s'ajuste horizontalement pour afficher la totalité du texte. Lorsque `WordWrap` a la valeur `True`, Visual Basic maintient la largeur du label, mais l'étend verticalement pour afficher autant de lignes que nécessaire.

La Figure 3.3 montre trois versions d'un même label, avec une propriété `Caption` identique, mais avec des combinaisons différentes des propriétés `AutoSize` et `WordWrap`.

Cela peut paraître étonnant, mais, pour que `WordWrap` fonctionne, `AutoSize` doit aussi avoir la valeur `True`. En effet, Visual Basic doit pouvoir étendre le label horizontalement, ne serait-ce qu'un petit peu, au cas où un même mot serait plus large que le label lui-même.

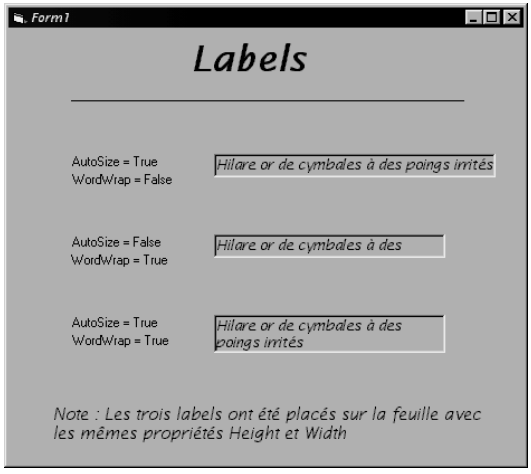


Figure 3.3
Les propriétés AutoSize et WordWrap affectent l'affichage du texte sur un label.

Le contrôle TextBox

On a recours aux zones de texte (*text boxes*) lorsque l'utilisateur doit taper quelque chose — réponse à une invite ou informations diverses. Il est souvent plus facile pour les utilisateurs de se voir proposer une valeur par défaut, ce que Visual Basic vous permet de faire. Si, par exemple, vous demandez à l'utilisateur de taper une date, vous pouvez afficher par défaut la date du jour, de sorte que l'utilisateur n'aura pas à la ressaisir.

Les zones de texte ne conviennent pas aux questions appelant des réponses du type oui/non. Pour inviter l'utilisateur à choisir entre deux valeurs ou à répondre par oui ou

par non, d'autres contrôles sont plus appropriés — comme les contrôles OptionButton (bouton d'option) ou ListBox (zone de liste déroulante), que nous étudierons de manière plus approfondie aux Chapitres 5 et 6.

Faire
A côté du contrôle TextBox, il est bon d'indiquer dans un label la valeur attendue. Pour une demande de nom et adresse, par exemple, placez avant le contrôle TextBox un label du type `Entrez votre nom :`*. Vos utilisateurs sauront ainsi précisément ce que l'on attend d'eux.*

Vous devez faire le distinguo entre la phase de création (designtime) et la phase d'exécution (runtime). La phase de création est la période pendant laquelle vous écrivez et maintenez une application. La phase d'exécution est celle où l'utilisateur exploite votre application. La valeur que vous définissez pour la propriété `Text` *d'une zone de texte lors de la phase de création sera la valeur par défaut pour l'utilisateur lors de la phase d'exécution. Cet utilisateur pourra alors modifier la valeur de la zone de texte, en la réécrivant ou en la modifiant.*

Pour paramétrer les zones de texte de votre feuille, vous vous servirez des propriétés suivantes :

- `Alignment`. Détermine l'alignement du texte dans une zone de texte dont la propriété `MultiLine` est définie comme `True`. La propriété `Alignment` est sans utilité pour une zone de texte qui ne contient qu'une ligne.
- `Locked`. Détermine si l'utilisateur peut, dans une zone de texte, entrer une nouvelle valeur ou modifier la valeur par défaut. Si la valeur de `Locked` est `True`, l'utilisateur ne peut intervenir sur le texte tant que le programme n'attribue pas, en cours d'exécution, la valeur `False`.
- `MaxLength`. Spécifie le nombre maximal de caractères que la zone de texte peut contenir. La valeur `0` indique une longeur illimitée.
- `MultiLine`. Lorsque sa valeur est `True`, cette propriété spécifie que la zone de texte peut recevoir plus d'une ligne de texte. Une barre de défilement verticale apparaît si l'utilisateur entre plus de texte que ne peut en contenir une seule ligne —à moins que vous n'ayez désactivé les barres de défilement avec la propriété `ScrollBars`. La propriété `Alignment`, dont la valeur par dégaut est `0-Left Justify`, détermine l'alignement du texte.
- `PasswordChar`. Désigne un caractère, comme l'astérisque, qui apparaîtra en lieu et place des caractères tapés par l'utilisateur dans la zone de texte. Ainsi, si l'utilisateur entre un mot de passe, seuls des astérisques apparaîtront à l'écran, de sorte qu'aucun

indiscret ne puisse le lire par-dessus son épaule. Bien que seuls des astérisques apparaissent, c'est le mot de passe réellement tapé par l'utilisation qui sera enregistré.

- `ScrollBars`. Détermine si des barres de défilement apparaîtront dans la zone de texte, et combien. Lorsque la valeur est `0-None`, aucune barre de défilement ne s'affiche. `1-Horizontal` n'autorise que les barres de défilement horizontales. `2-Vertical` n'autorise que les barres de défilement verticales. `3-Both` affiche des barres de défilement verticales et horizontales.

Pour que des barres de défilement apparaissent dans la zone de texte, la propriété `MultiLine` doit avoir la valeur `True`.

- `Text`. Spécifie le texte qui apparaît par défaut dans la zone de texte.

Les zones de texte et les champs de formulaires appellent un curseur spécial. Votre utilisateur saisit et modifie le texte à l'aide d'un *curseur de texte*. Lorsqu'on clique sur le contenu d'une zone de texte, le curseur de texte s'active pour en permettre la modification.

Le curseur de texte, également nommé point d'insertion, est une barre verticale pour saisir ou modifier du texte dans des contrôles tels que les zones de texte.

Chargez et exécutez l'application Controls depuis le dossier `Samples\VB98\Controls` de MSDN. Sélectionnez l'option TextBox pour vous entraîner un peu. L'écran qui s'affiche est reproduit en Figure 3.4. Vous pourrez ainsi vous familiariser avec les zones de texte monolignes et multilignes, ainsi qu'avec les curseurs de texte et les barres de défilement. L'application Controls démontre également que l'on peut effectuer dans et depuis une zone de texte les opérations Windows standards de copier, couper et coller.

Le contrôle CommandButton

Les applications contiennent presque toujours des boutons de commande. Les boutons de commande permettent à l'utilisateur d'indiquer qu'une réponse est prête, que l'imprimante est rechargée en papier, ou qu'il souhaite quitter le programme. Comme nous l'avons vu au chapitre précédent, Visual Basic prend en charge l'animation du bouton qui signale graphiquement le clic.

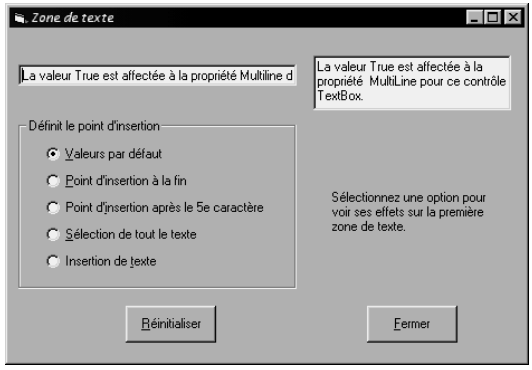


Figure 3.4
L'application exemple Controls met en œuvre les divers types de zones de texte.

Une touche de raccourci est une touche qui, combinée à la touche Alt, déclenche une réponse du programme. Par exemple, les options de barre de menus ont toutes une touche de raccourci : Alt-F pour le menu Fichier, etc. Dans une option de barre de menus ou dans le label d'un bouton de commande, la lettre soulignée indique la touche de raccourci correspondante.

Les boutons de commande ne servent pas qu'aux clics de souris. Vous pouvez assigner au label du bouton de commande une touche de raccourci. L'utilisateur pourra ainsi enclencher le bouton de commande en appuyant sur la combinaison de touches correspondantes, telle que Alt-R. C'est dans la propriété `Caption` *du bouton que la touche de raccourci est définie. Outre les touches de raccourci, l'utilisateur peut enclencher le bouton de commande en appuyant sur Entrée ; il faut pour cela que le bouton soit le contrôle sélectionné ou que sa propriété* `Default` *ait la valeur* `True` *(voyez la section sur le focus pour plus d'informations). Vous pouvez enfin attribuer à un bouton de commande la touche Echap, par exemple pour les boutons du type Quitter ou Annuler (dans ce dernier cas, c'est la propriété* `Cancel` *qui gère la touche Echap).*

Voici quelques propriétés utiles pour la programmation des boutons de commande :

- `Cancel`. Détermine la réaction du bouton de commande à la touche Echap. Si la valeur est `True`, l'utilisateur peut enclencher le bouton de commande par la touche Echap, exactement comme s'il avait cliqué. Un seul bouton sur la feuille peut avoir la propriété `Cancel` définie comme `True`. Si vous spécifiez cette valeur de propriété pour plus d'un bouton, Visual Basic ne retient que le dernier, définissant tous les autres comme `False`.

- `Caption`. Contient le texte qui apparaîtra sur le bouton de commande. Pour définir une lettre comme touche de raccourci, il suffit de la faire précéder d'une esperluette (&). Ainsi, une propriété `Caption` de valeur `&Quitter` donnera un bouton comme celui qui est représenté en Figure 3.5. L'utilisateur pourra, au choix, enclencher ce bouton en cliquant dessus ou en appuyant sur Alt-Q.

Figure 3.5
On définit une touche de raccourci en plaçant, dans la valeur de Caption, une esperluette `&` avant la lettre choisie (la touche de raccourci est soulignée sur le bouton).

- `Default`. Détermine la réaction du bouton de commande à la touche Entrée. Si la valeur est `True`, l'utilisateur peut enclencher le bouton en appuyant sur Entrée — à moins qu'il n'ait mis le focus sur un autre contrôle (voir section suivante). Si vous spécifiez cette valeur de propriété pour plus d'un bouton, Visual Basic ne retient que le dernier, définissant tous les autres comme `False`. Le bouton dont la propriété `Default` a pour valeur `True` est sélectionné par défaut à l'affichage de la feuille.
- `Picture`. Spécifie l'image qui apparaîtra sur le bouton à la place d'un libellé. Il faut pour cela que la propriété `Style` ait été préalablement définie comme `1-Graphical`.
- `Style`. Cette propriété détermine si le bouton doit afficher un libellé (valeur `0-Standard`) ou une image (valeur `1-Graphical`).

Prenez garde d'attribuer la même touche de raccourci à plus d'un contrôle. Si tel était le cas, Visual Basic ne retiendrait que le premier contrôle dans l'ordre du focus (voir section suivante), et la touche de raccourci deviendrait inopérante pour les autres contrôles.

Le contrôle Image

Comme vous avez placé une image sur l'application créée au chapitre précédent, une brève présentation du contrôle Image est de rigueur ici. Le contrôle Image est, avec le contrôle PictureBox, l'un des contrôles qui affichent des images. Les propriétés de ce contrôle déterminent le fichier graphique à utiliser, et spécifient si la taille du contrôle doit s'ajuster à celle du fichier, ou si la taille du fichier doit s'ajuster à celle du contrôle. Vous en apprendrez plus sur les contrôles Image et PictureBox au Chapitre 14.

Visual Basic propose, bien sûr, beaucoup d'autres contrôles, que vous découvrirez au fil de votre apprentissage. Les sections suivantes décrivent de quelle manière l'utilisateur enclenche un contrôle par l'intermédiaire du clavier.

Le focus

Sur une même feuille, un seul contrôle peut *avoir le focus* à la fois. L'ordre dans lequel les contrôles auront le focus dépend de celui dans lequel vous les avez placés sur la feuille, ou plus exactement dans l'ordre spécifié dans la propriété `TabIndex` de chaque contrôle. Tous les contrôles ne peuvent pas avoir le focus. C'est un privilège réservé aux contrôles qui appellent une interaction avec l'utilisateur. Ainsi, un contrôle Label ne pourra pas avoir le focus, car l'utilisateur ne peut en aucun cas intervenir sur son contenu.

Le focus, ou focus de contrôle, désigne le contrôle actuellement sélectionné. Visual Basic signale le focus en mettant le contrôle en surbrillance, ou en l'entourant d'un cadre pointillé.

Notez que la touche de raccourci pour le troisième bouton est Alt-A, et non Alt-M. La touche M étant déjà attribuée au bouton du milieu (sa propriété Caption a pour valeur `&Moyen`), le programmeur a dû en attribuer une autre au troisième bouton.

Rappelez-vous que le contrôle pour lequel la propriété `Default` a pour valeur `True` aura automatiquement le focus à l'affichage de la feuille. Naturellement, l'utilisateur peut toujours mettre le focus sur un autre contrôle.

Examinez la Figure 3.6. Le bouton de commande du milieu se distingue par le cadre pointillé qui l'entoure. C'est généralement de cette manière que Visual Basic signale le contrôle qui a le focus. Là, si l'utilisateur appuie sur Entrée, le bouton de commande central s'enclenche, car il a le focus. L'utilisateur peut aussi bien, en appuyant sur Tab ou sur Maj-Tab, mettre le focus sur un autre contrôle. Si donc il appuie sur Tab, le focus passe au troisième bouton, qui peut alors s'enclencher en réaction à la touche Entrée.

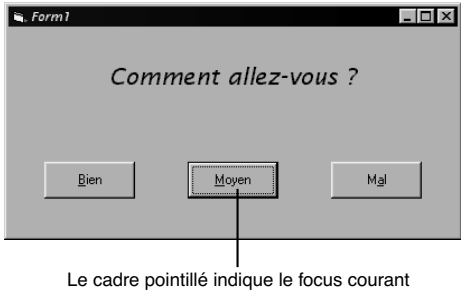


Figure 3.6
Le contrôle encadré est celui qui a le focus. L'utilisateur peut, aussi bien, mettre le focus sur un autre contrôle avant d'appuyer sur Entrée.

Tous les contrôles ont une propriété `TabIndex`. A mesure que vous disposez les contrôles sur la feuille, Visual Basic définit automatiquement la propriété `TabIndex` comme `0`, `1`, etc., attribuant un nombre unique et séquentiel à la propriété `TabIndex` de chaque contrôle. Même les contrôles qui, d'ordinaire, n'appellent pas d'interaction avec l'utilisateur, tels que les labels, ont une propriété `TabIndex`. La propriété `TabIndex` définit l'ordre du focus.

Vous ne placerez pas toujours les contrôles en suivant l'ordre des valeurs de `TabIndex`. Il vous arrivera notamment de placer un contrôle entre deux autres. Il faudra peut-être modifier l'ordre de `TabIndex` pour maintenir l'ordre de focus que vous désirez. Par exemple, vous pouvez choisir de déplacer le focus selon l'alignement vertical des contrôles, ou selon leur alignement horizontal ; en fonction de votre choix, la touche Tab provoquera un déplacement du focus de haut en bas ou de gauche à droite. Cet ordre de déplacement du focus est spécifié par les valeurs de la propriété `TabIndex` pour chaque contrôle.

Si le prochain contrôle à recevoir le focus — selon les valeurs de `TabIndex` — est un label, Visual Basic passe le focus au contrôle suivant. Vous pouvez, en conséquence, proposer à vos utilisateurs une touche de raccourci pour une zone de texte. Examinez la Figure 3.7. Le label `Nom :` a une propriété `TabIndex` de valeur plus grande que la zone de texte suivante. Nonobstant le contrôle qui a le focus, lorsque l'utilisateur appuie sur Alt-F, Visual Basic passe le focus au label, qui le repasse immédiatement au contrôle suivant (la zone de texte) parce que les labels ne peuvent avoir le focus. Ainsi, les touches de raccourci que vous attribuez aux labels identifient, en fait, les zones de texte, et permettent aux utilisateurs d'en modifier le contenu. Pour cela, il faut naturellement que les propriétés `TabIndex` de chaque paire label/zone de texte aient des valeurs consécutives.

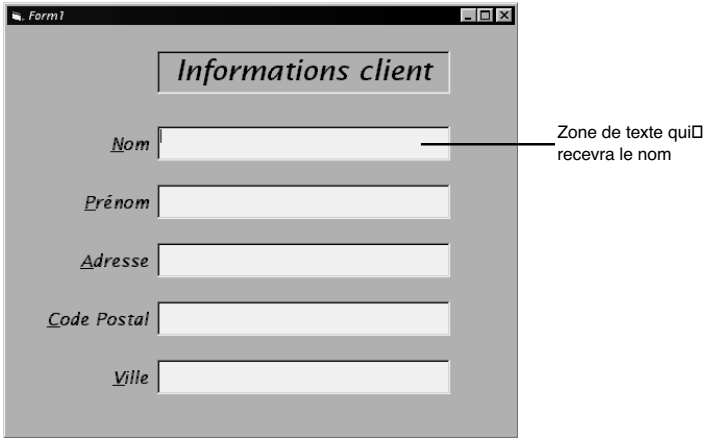


Figure 3.7
L'utilisateur peut appuyer sur Alt-N pour entrer son nom dans la zone de texte.

Les procédures événementielles

Les procédures événementielles sont parfois d'un abord difficile pour le programmeur débutant ; pourtant, le principe en est fort simple. Lorsque l'utilisateur appuie sur un bouton de commande ou tape dans une zone de texte, quelque chose doit en avertir l'application. Comme nous l'avons vu dans le chapitre précédent, Windows reçoit des événements de plusieurs sources différentes. La plupart des événements viennent directement de l'utilisateur qui, *via* le clavier ou la souris, interagit avec une application en cours d'exécution.

Si un événement est déclenché et qu'il ne s'agisse pas d'un événement système, tel qu'un clic sur le bouton Démarrer, Windows passe l'événement à l'application. Si vous avez écrit une procédure événementielle pour traiter cet événement précis, l'application y répond. Si vous n'avez pas écrit une telle procédure événementielle, l'événement n'est pas traité et, pour ainsi dire, se perd.

Toutes sortes d'événements peuvent se produire : clic, double-clic, frappe au clavier, etc. En outre, plusieurs contrôles de la feuille peuvent répondre aux mêmes types d'événements. Par exemple, un bouton de commande et une zone de texte peuvent tous deux recevoir un événement `Click`, l'utilisateur pouvant cliquer sur l'un comme sur l'autre. Il s'ensuit que vous ne devez pas seulement écrire une procédure événementielle pour un événement particulier, mais aussi spécifier le contrôle auquel cet événement revient.

Pour votre compréhension générale de Visual Basic, il est capital de saisir cette nécessité d'écrire des procédures événementielles pour tous les événements et tous les contrôles. Combien de procédures événementielles `Click` *faudra-t-il écrire pour gérer les trois boutons de commande d'une feuille ? Il faudra en écrire trois, puisque l'utilisateur peut cliquer sur chacun des trois boutons.*

Si vous écrivez une procédure `Click` *sans l'affecter à un contrôle particulier, le programme ne pourra traiter de façon spécifique les différents boutons de commande. Une procédure* `Click` *séparée doit être écrite pour chaque bouton de commande. Lorsque l'événement* `Click` *est passé à votre programme, Windows passe également le contrôle qui a généré cet événement. Pour que votre application puisse répondre à l'événement, vous devez avoir écrit une procédure événementielle pour le contrôle et pour l'événement.*

Imaginons que votre application affiche quatre boutons de commande sur une feuille. Voici ce qui se passe :

1. Lorsque l'utilisateur clique sur l'un des boutons, Windows reconnaît qu'un événement vient de se produire.
2. Windows analyse cet événement et constate qu'il relève de votre application.
3. Windows passe l'environnement et le contrôle à votre application.
4. Si votre application dispose d'une procédure pour le contrôle qui a reçu l'événement, le code de cette procédure (que vous aurez écrit dans la fenêtre Code) s'exécute.

Le code de la procédure événementielle s'exécutera si et seulement si l'événement correspondant a lieu. Cette qualité fait de Visual Basic un système très réactif : vous écrivez toutes sortes de procédures événementielles qui attendront sagement qu'un événement précis les appelle ; c'est à ce moment qu'elle se mettront au travail, indépendamment de toute autre activité du programme.

Les événements des contrôles courants

Nous vous présentons ici les événement les plus courants, liés aux contrôles que vous connaissez déjà. Les feuilles comme les contrôles peuvent recevoir des événements. Voici quelques événements de feuilles susceptibles de se produire lors de l'exécution :

- `Activate`. Se produit lorsqu'une feuille a le focus. Dans une application qui contient plusieurs feuilles, l'événement `Activate` a lieu quand l'utilisateur a une feuille différente en cliquant dessus ou en la sélectionnant depuis un menu.
- `Click`. Se produit lorsque l'utilisateur clique quelque part sur la feuille. Si l'utilisateur clique sur une feuille partiellement cachée par la feuille qui a le focus, les deux événements, `Click` et `Activate`, se produisent.

Lorsque l'utilisateur clique du bouton droit sur la feuille, c'est également un événement `Click` qui se déclenche. En progressant dans votre apprentissage, vous apprendrez à déterminer lequel des deux boutons de la souris a été employé pour le clic.

- `DblClick`. Se produit lorsque l'utilisateur double-clique sur la feuille.
- `Deactivate`. Se produit lorsque le focus passe à une autre feuille. En fait, lorsque l'utilisateur sélectionne une autre feuille, il se produit à la fois un événement `Activate` et un événement `Deactivate`. Vous avez le choix d'écrire une procédure événemen-

tielle pour les deux événements sur chaque feuille, pour un seul des deux événements sur l'une des deux feuilles, ou une combinaison des deux, selon les besoins de votre application.

- `Initialize`. Se produit lorsqu'une feuille est générée pour la première fois.
- `Load`. Se produit au moment précis où la feuile est chargée dans la mémoire active et s'affiche à l'écran.
- `Paint`. Se produit lorsque Windows doit, suite à une action de l'utilisateur, redessiner la feuille pour faire apparaître une partie de la feuille qui était cachée par un autre objet, tel qu'une icône.
- `Resize`. Se produit lorsque l'utilisateur change la taille de la feuille.
- `Unload`. Se produit lorsque l'application provoque, par exécution d'une partie du code, le désaffichage d'une feuille. Lorsqu'une application se termine, toutes les feuilles chargées sont déchargées ; vous devrez donc écrire une procédure événementielle `Unload` pour chaque feuille si vous souhaitez que la fermeture de l'application s'accompagne d'une sauvegarde des fichiers et d'un "nettoyage" de la mémoire et de l'écran.

Voici les événements les plus courants pour les contrôles TextBox :

- `Change`. Se produit lorsque l'utilisateur modifie le texte.
- `Click`. Se produit lorsque l'utilisateur clique sur la zone de texte.
- `DblClick`. Se produit lorsque l'utilisateur double-clique sur la zone de texte.

Rappelez-vous qu'il existe beaucoup plus de contrôles que vous n'en découvrez aujourd'hui. Des événements disponibles pour la plupart des contrôles, tels que les événements liés à la souris et au clavier, seront traités plus loin dans ce livre. Nous nous contentons pour l'instant des événements qui vous seront les plus utiles lors de vos premiers développements.

La plupart de ces événements de zones de texte s'appliquent également aux labels. Mais les labels, de par leur nature, déclenchent les événements d'une manière légèrement différente. Par exemple, le contrôle Label supporte l'événement `Change`, même si l'utilisateur ne peut intervenir directement sur un label. C'est que le code Visual Basic peut, lui, modifier un label ; et, lorsque cela se produit, un événement `Change` a lieu.

Le contrôle Image supporte le même jeu d'événements que le contrôle Label. Les deux sont en effet très semblables, à cela près que le contrôle Image affiche une image au lieu de texte.

Le contrôle CommandButton supporte les mêmes événements que le contrôle TextBox.

En programmant les événements de boutons de commande, gardez ceci à l'esprit :

- Dans le cas où une feuille contient un seul bouton de commande, la frappe de la barre d'espace, lorsque le bouton a le focus, déclenche la procédure événementielle dudit bouton.
- Si la propriété `Cancel` d'un bouton de commande est définie comme `True`, la frappe de la touche Echap déclenche l'événement `Click`.
- La frappe des touches de raccourci peut déclencher l'événement `Click` d'un bouton de commande.

Tous les événements d'une application ne viennent pas forcément d'une action de l'utilisateur. Les événements peuvent être déclenchés depuis le code Visual Basic. Par exemple, vous demandez à l'utilisateur de cliquer sur un bouton de commande lorsqu'il est prêt à visualiser le résultat d'un calcul. La procédure événementielle `Click` du bouton de commande provoque le calcul et l'affichage de ce résultat. Mais votre code stipule aussi que, au-delà d'un certain temps, le même événement `Click` soit automatiquement déclenché. L'événement peut donc se produire, et le résultat apparaître, avec ou sans l'intervention de l'utilisateur.

Ecrire des procédures événementielles

Les procédures événementielles sont constituées de code Visual Basic. Plus précisément, les procédures événementielles sont des sections de code chargées de gérer un événement d'un contrôle particulier. Un même contrôle peut avoir plusieurs procédures événementielles selon qu'il doive répondre à différents types d'événements.

Le nom de la procédure événementielle permet à Visual Basic de déterminer deux choses :

- Quel contrôle déclenchera la procédure.
- Quel événement déclenchera la procédure.

Voici le format de tous les noms de procédures événementielles :

```
ControlName_EventName ( )
```

L'underscore, ou caractère de soulignement, sépare le nom du contrôle de celui de l'événement ; il est indispensable. Toutes les procédures événementielles sont nommées de

cette manière. Selon, ce format, une procédure nommée `cmdExit_DblClick ()` s'exécutera *si et seulement si* l'événement `DblClick` du bouton de commande `cmdExit` se produit.

Pour ce qui est des parenthèses, vous apprendrez à y mettre les valeurs adéquates en progressant dans votre apprentissage. Même quand elles sont vides, comme dans l'application du chapitre précédent, les parenthèses sont tout aussi indispensables que l'underscore. Elles permettent notamment de distinguer les noms de procédures événementielles des noms de contrôles — même si elles ne font pas partie du nom lui-même.

Le code contenu dans la procédure événementielle `cmdExit_DblClick ()` ne s'exécute que si l'utilisateur double-clique sur le bouton de commande `cmdExit`. Si ce devait être la seule procédure événementielle que contienne le code, l'application ignorerait tout autre événement que `DblClick` sur `cmdExit`. Si, par exemple, l'utilisateur se contente de cliquer sur le bouton, rien ne se passe : la procédure événementielle attend un *double*-clic.

La plupart des procédures événementielles que vous écrirez lors de vos premiers pas en Visual Basic commenceront par les mots `Private Sub`. Le mot clé `Private` est toutefois optionnel ; si vous ne le spécifiez pas, Visual Basic considère par défaut que la procédure est "privée".

Visual Basic supporte deux types de procédures : les *fonctions* et les *sous-routines*. Les procédures événementielles sont toutes des sous-routines. Le corps d'une procédure événementielle peut être de plusieurs centaines de lignes, quoiqu'il faille toujours écrire aussi court que possible. Si vous en venez à écrire une procédure exceptionnellement longue, essayez de la fractionner en procédures plus petites ; la maintenance du programme n'en sera que plus facile. Le Listing 3.1 donne un exemple de ce à quoi ressemblerait `cmdExit_DblClick ()` dans une application.

Ces nombres qui commencent chaque ligne du Listing 3.1, vous les retrouverez toujours dans la suite de l'ouvrage. Ces nombres ne font pas partie du programme ; il ne s'agit que d'une numérotation de référence qui nous permettra, dans les leçons, de vous renvoyer à une ligne précise et immédiatement identifiable.

Listing 3.1 : Une procédure événementielle qui s'exécute lorsque l'utilisateur double-clique sur le bouton de commande

```
1: Private Sub cmdExit_DblClick ( )
2:    lblTitle.Caption = "Nouvelle page"
3:    intTotal = intCustNum + 1
4: End Sub
```

La fonction est un peu comme le lanceur au base-ball : elle ne fait qu'envoyer une valeur, dite valeur renvoyée, vers un autre coin du programme. Le mot clé `Function` *distingue ce type de procédures des sous-routines. La sous-routine, indiquée par le mot clé* `Sub`*, ne renvoie pas de valeur, mais effectue une tâche précise à travers le code. Rappelons-le : les procédures événementielles sont toutes des sous-routines. Vous saisirez mieux la distinction entre fonctions et sous-routines en progressant dans vos exercices.*

La première ligne de ce listing en dit beaucoup sur la procédure événementielle. Elle indique d'abord que la procédure est privée, c'est-à-dire uniquement accessible au module d'application courant. Elle indique également qu'il s'agit d'une sous-routine, et donc qu'aucune valeur n'en sera renvoyée. D'après le nom lui-même, on sait que le contrôle concerné est le bouton de commande (désigné par le préfixe `cmd`) que le programmeur a nommé `cmdExit`. On sait enfin que la procédure événementielle ne répond qu'aux doubles-clics sur ce bouton de commande.

Le corps de la procédure événementielle n'occupe que deux lignes. Vous n'avez pas à vous en soucier pour l'instant. La dernière ligne du code referme la procédure et vous permet, ainsi qu'à Visual Basic, de savoir où elle finit. (Les fonctions, quant à elles, se terminent toutes par l'instruction `End Function`.)

C'est dans le fenêtre Code que vous tapez tout le code. La fenêtre Code fonctionne comme un simple traitement de texte. Lorsque vous êtes prêt à écrire votre procédure événementielle, vous pouvez accéder à la fenêtre Code de plusieurs manières : en sélectionnant le contrôle auquel la procédure est destinée, puis en choisissant Affichage, Code ; en cliquant sur le bouton Afficher l'objet de la fenêtre Propriétés. Mais la méthode la plus simple, c'est de double-cliquer sur le contrôle. Visual Basic ouvre automatiquement la fenêtre Code pour cet objet, anticipe sur vos intentions en vous proposant les événements le plus couramment attribués, et écrit même la première et la dernière ligne de la procédure pour vous ! C'est précisément ce qui s'est passé dans l'application du chapitre précédent, lorsque vous avez double-cliqué sur le bouton de commande. Estimant que l'événement le plus probable pour un tel contrôle est l'événement `Click`, Visual Basic a affiché dans la fenêtre Code ces deux lignes :

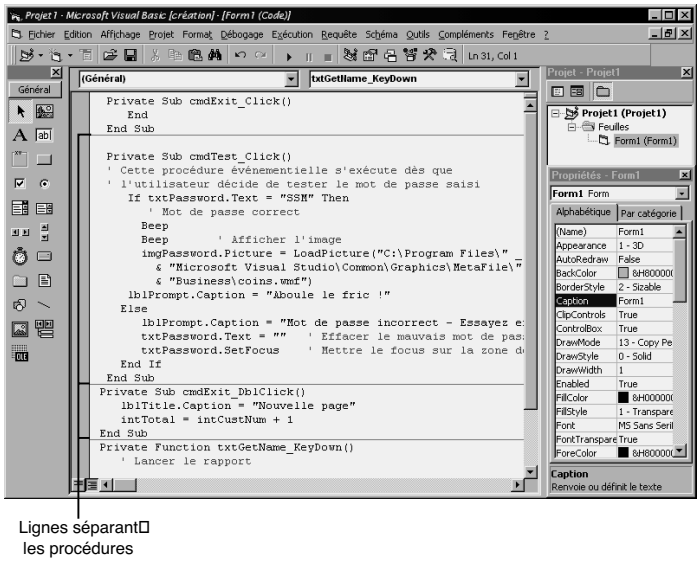
```
Private Sub cmdHappy_Click ()
End Sub
```

Visual Basic a même placé le curseur d'insertion entre les deux lignes, de sorte qu'il ne vous reste qu'à taper le corps de la procédure. Lorsque vous en avez terminé, vous pouvez écrire en dessous une autre procédure événementielle (auquel cas, vous devrez taper vous-même la première et la dernière ligne), ou cliquer de nouveau sur le bouton Afficher l'objet de la fenêtre Propriétés pour revenir à la fenêtre Feuilles.

Si l'anticipation de Visual Basic était fausse, et que vous souhaitiez écrire une procédure événementielle différente de celle qui vous est proposée, il suffit de changer le nom, par exemple pour `cmdHappy_DblClick ()`*, et de compléter le reste.*

La Figure 3.8 montre une fenêtre Code affichant le code de plusieurs procédures événementielles. La fenêtre Code sépare les procédures, et supporte les opérations Windows standards de copier, couper. En revanche, et contrairement à un vrai traitement de texte, la fenêtre Code ne génère pas de retours à la ligne automatiques. La raison en est que chaque instruction Visual Basic doit constituer une ligne propre. Une ligne extraordinairement longue peut être continuée à la ligne suivante, à condition de placer un underscore à la fin de la première ligne Ce signe indique à Visual Basic que les deux lignes doivent être traitées comme une seule ; vous, de votre côté, vous pouvez visualiser la totalité de l'instruction sans recourir à la barre de défilement.

Figure 3.8
La fenêtre Code fonctionne comme un traitement de texte pour procédures.



Les projets bonus

Vous avez eu droit à beaucoup de théorie. Si ce chapitre a répondu à certaines de vos questions, il en a sans doute soulevé de nouvelles. Pour mettre les choses en perspective, vous trouverez, entre ce chapitre et le suivant, une section spéciale qui propose un projet bonus sur les contrôles, propriétés et événements. Ce projet vous invite à créer de nouveau une application à partir de zéro afin de mettre en pratique les enseignements de ce chapitre. Vous n'y rencontrerez pas d'instructions aussi détaillées que dans l'application du chapitre précédent, parce que vous êtes maintenant familiarisé avec le fonctionnement de Visual Basic. D'autres projets bonus vous seront proposés, entre certains chapitres de ce livre, afin d'affermir les connaissances acquises. Considérez-les comme des devoirs à faire pour le jour suivant.

En résumé

Ce chapitre était plus théorique que les deux premiers. Vous devriez maintenant être en mesure de mieux comprendre les contrôles, les propriétés et les événements. La nature des procédures devrait également vous apparaître plus clairement, même s'il vous reste à peu près tout à apprendre du code Visual Basic. Vous avez retenu qu'une procédure événementielle doit exister pour chacun des contrôles et des événements auxquels vous souhaiter faire réagir votre programme. Sans procédure événementielle adéquate, l'événement serait ignoré.

Le chapitre suivant vous enseigne à ajouter des menus à votre application afin d'en faciliter l'utilisation.

Questions et réponses

Q Pourquoi faut-il indenter le corps des procédures événementielles ?

R En fait, vous devez indenter le corps de *toutes* les procédures. L'indentation n'est pas obligatoire en soi, mais elle vous permet de distinguer les événements les uns des autres dans les longues lignes de code. La fenêtre Code vous facilite déjà la tâche en séparant les procédures par des lignes. Mais, lorsque vous imprimerez un listing pour l'étudier, l'indentation vous permettra de repérer les procédures individuelles.

Q Puis-je inventer des noms de procédures événementielles ?

R La seule chose que vous puissiez changer dans le nom d'une procédure événementielle, c'est le nom du contrôle qui déclenche la procédure. Rappelez-vous que le format spécifique employé dans les noms de procédures événementielles permet à Visual Basic de déterminer les contrôles et les événements censés déclencher la procédure. Vous êtes libre de nommer comme bon vous semble les autres types de procédures, sous-routines ou fonctions, mais vous devrez toujours respecter, pour les procédures événementielles, le modèle de noms présenté aujourd'hui.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous assurerez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous amèneront à mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Qu'est-ce qu'une touche de raccourci ?
2. Les propriétés supportent de multiples événements. Vrai ou faux ?
3. Pourquoi affecter l'événement `Cancel` à un bouton de commande ?
4. Comment reconnaît-on le contrôle qui a le focus ?
5. Comment l'utilisateur déplace-t-il le focus d'un contrôle à l'autre ?
6. Quelle propriété définit l'ordre du focus ?
7. Devinette : `LoadPicture ()` est-ce une sous-routine, une fonction ou une procédure événementielle ?
8. Lorsque vous double-cliquez sur un contrôle dans la fenêtre Feuilles, Visual Basic génère automatiquement la première et la dernière ligne de la procédure événementielle `Click`. Vrai ou faux ?
9. On peut déclencher un événement, tel que `DblClick`, depuis le code Visual Basic. Vrai ou faux ?
10. A quoi sert la propriété `PasswordChar` ?

Exercices

1. Ecrivez la première ligne de la procédure événementielle `Load` pour une feuille nommée `frmMyApp`.
2. **Chasse au bogue** : pourquoi les lignes qui suivent ne forment-elles pas une procédure événementielle ?

```
1: Private Function txtGetName_KeyDown ()
2:     ' Lancer le rapport
3:    Call ReportPrint
4:   lblWarning.Caption = "Tenez-vous prêt…"
5: End Function
```

3. Créez une application contenant trois zones de texte multilignes. Donnez à ces zones de texte une hauteur suffisante pour pouvoir afficher trois ou quatre lignes de texte. Intégrez une barre de défilement horizontale à la première zone de texte, une barre de défilement verticale à la deuxième, et les deux types de barres de défilement à la troisième. Pour chacune des trois zones de texte, spécifiez le contenu par défaut `Tapez ici`. Ajoutez à votre application un bouton de commande Quitter qui permette à l'utilisateur de fermer le programme en appuyant sur Alt-Q.

PB1

Contrôles, propriétés et événements

Intercalés çà et là entre les chapitres de ce livre, les sections "Projet bonus" vous aideront à approfondir et à mettre en pratique les connaissances acquises. Vous y trouverez des listings d'applications complètes. Prenez le temps de créer ces applications ; elles vous familiariseront avec l'environnement Visual Basic et vous feront avancer dans la maîtrise des techniques de programmation.

L'application de ce premier projet bonus met en œuvre la propriété de zones de texte `PasswordChar`. Le programme se sert de cette propriété pour demander et valider un mot de passe. Si l'utilisateur entre un mot de passe correct, une image apparaît.

Faites des expériences sur les programmes des projets bonus. A mesure que vous progresserez, vous pourrez modifier ces applications pour vérifier une supposition ou vous exercer à l'écriture de code. Vous ne comprendrez sans doute pas tout du code à saisir pour cette application. Mais rien ne presse, et vous serez bientôt à même de l'analyser.

Le mot de passe pour cette application est `SSM`. Mais ne le dites à personne…

Les éléments visuels

La Figure PB1.1 montre la fenêtre Feuilles de l'application que vous allez créer. Vous savez déjà placer des contrôles sur une feuille. Le Tableau PB1.1 fournit tous les contrôles et les propriétés nécessaires. Suivez, pour commencer, les étapes classiques de la création d'application :

1. Sélectionnez Fichier, Nouveau projet.
2. Sélectionnez l'icône EXE standard.
3. Donnez aux propriétés de la feuille les valeurs listées dans le Tableau PB1.1
4. Placez sur la feuille les contrôles mentionnés et définissez leurs propriétés. Pour toutes les propriétés qui n'apparaissent pas dans le tableau, laissez les valeurs par défaut.

Figure PB1.1
Cette application utilise la propriété PasswordChar du contrôle zone de texte.

Vous n'êtes pas obligé de définir les propriétés d'un contrôle dès qu'il est placé sur la feuille. Vous pouvez commencer par placer tous les contrôles, pour ensuite définir les propriétés de chacun.

Tableau PB1.1 : Contrôles et propriétés à utiliser pour l'application

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmPassword`
Feuille	`Caption`	`Essayez un mot de passe`

Tableau PB1.1 : Contrôles et propriétés à utiliser pour l'application *(suite)*

Contrôle	*Propriété*	*Valeur*
Feuille	`Height`	`5610`
Feuille	`Width`	`8475`
Image	`Name`	`imgPassword`
Image	`BorderStyle`	`1-Fixed Single`
Image	`Height`	`1890`
Image	`Left`	`3000`
Image	`Stretch`	`True`
Image	`Top`	`2640`
Image	`Width`	`2295`
Label	`Name`	`lblPrompt`
Label	`BorderStyle`	`1-Fixed Single`
Label	`Caption`	`Tapez votre mot de passe ici`
Label	`Font`	`MS Sans Serif`
Label	`Font Size`	`14`
Label	`Font Style`	`Bold`
Label	`Height`	`855`
Label	`Left`	`2520`
Label	`Top`	`600`
Label	`Width`	`3375`
Zone de texte	`Name`	`txtPassword`
Zone de texte	`Height`	`375`
Zone de texte	`Left`	`3360`
Zone de texte	`PasswordChar`	`*`

Tableau PB1.1 : Contrôles et propriétés à utiliser pour l'application *(suite)*

Contrôle	*Propriété*	*Valeur*
Zone de texte	Text	(Laissez vide en effaçant la valeur par défaut)
Zone de texte	`Top`	`1800`
Zone de texte	`Width`	`1695`
Bouton de commande	`Name`	`cmdTest`
Bouton de commande	`Caption`	`&Tester le mot de passe`
Bouton de commande	`Left`	`6360`
Bouton de commande	`Top`	`3000`
Bouton de commande #2	`Name`	`cmdExit`
Bouton de commande #2	`Caption`	`E&xit`
Bouton de commande #2	`Left`	`6360`
Bouton de commande #2	`Top`	`3720`

Ajouter le code

Une fois créée la fenêtre Feuilles, vous êtes prêt à entrer le code. Le Listing PB1.1 fournit ce code. Notez l'existence de deux procédures événementielles : `cmdExit_Click()` et `cmdTest_Click()`. Chacune de ces deux procédures répond à l'événement `Click` pour un bouton particulier. Lorsque l'utilisateur clique sur le bouton de commande `cmdExit`, `cmdExit_Click()` s'exécute. Lorsque l'utilisateur clique sur le bouton de commande `cmdTest`, `cmdTest_Click()` s'exécute.

`Click` est l'événement le plus courant pour un bouton de commande. Vous pouvez ouvrir rapidement une procédure événementielle pour chaque bouton de commande et laisser Visual Basic remplir la première et la dernière ligne. Pour saisir le corps (qui tiendra en une ligne) de la procédure événementielle `Click` du bouton de commande `cmdExit`, double-cliquez sur ce dernier dans la fenêtre Feuilles, puis saisissez la ligne de code. Procédez de même pour le bouton de commande `cmdTest`.

Listing PB1.1 : Ce code active la feuille de mot de passe

```
1: Private Sub cmdExit_Click()
2:    End
3: End Sub
4:
5: Private Sub cmdTest_Click()
6: ' Cette procédure événementielle s'exécute dès que
7: ' l'utilisateur décide de tester le mot de passe saisi
8:    If txtPassword.Text = "SSM" Then
9:       ' Mot de passe correct
10:      Beep
11:      Beep       ' Affiche l'image
12:      imgPassword.Picture = LoadPicture("C:\Program Files\" _
13:        & "Microsoft Visual Studio\Common\Graphics
         ➥\MetaFile\" _
14:        & "Business\coins.wmf")
15:    lblPrompt.Caption = "Aboule le fric !"
16:   Else
17:      lblPrompt.Caption = "Mot de passe incorrect -
       ➥Essayez encore "
18:      txtPassword.Text = ""    ' Efface le mauvais mot de passe
19:      txtPassword.SetFocus     ' Met le focus sur la zone de texte
20:   End If
21: End Sub
```

Analyse

Naturellement, vous devez encore apprendre à déchiffrer les instructions de programmation Visual Basic. Mais la petite explication du programme que nous vous proposons ici constitue une excellente introduction au Chapitre 5, qui expose les fondements du code.

Les lignes 1 à 3 constituent la procédure événementielle `Click` du bouton `cmdExit`. Lorsque l'utilisateur clique sur le bouton `cmdExit`, l'instruction `End` provoque la fermeture de l'application. L'application reste affichée à l'écran, même si l'utilisateur a entré le bon mot de passe, et jusqu'à ce qu'il clique sur le bouton de commande `cmdExit` (ou qu'il ferme la fenêtre du programme).

En tapant le code, vous constaterez que Visual Basic utilise des couleurs différentes selon le texte. Cette coloration de syntaxe est une fonction utile, notamment pour localiser les bogues assez tôt dans la phase d'écriture. Plus vous programmerez, mieux vous reconnaîtrez les couleurs spécifiques. Les mots clés Visual Basic, tels que les commandes, sont toujours affichés en bleu. Les objets, tels que contrôles et propriétés, sont toujours affichés en noir. Les autres éléments de code apparaisent en vert. Ainsi, si un mot clé apparaît en vert, c'est que Visual Basic ne le reconnaît pas, et vous savez

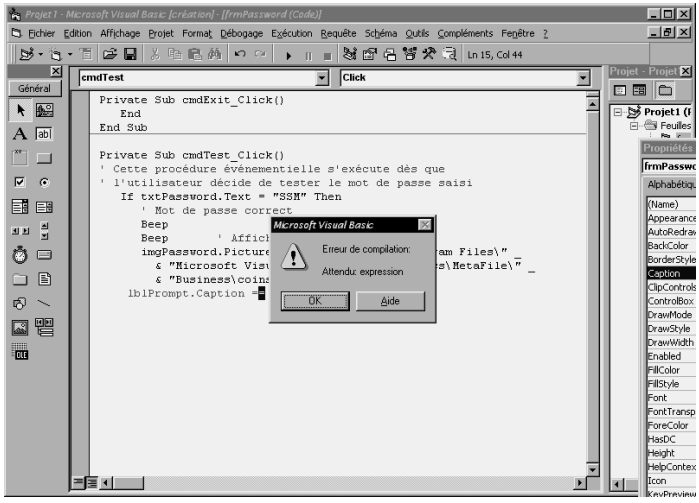
immédiatement que vous avez tapé quelque chose d'incorrect. Le jeu de couleurs peut être modifié depuis la page Format de l'éditeur du menu Outils, Options.

La *syntaxe* est l'ensemble des règles d'écriture d'un langage de programmation. Si vous orthographiez mal une commande ou oubliez un signe de ponctuation, une erreur de syntaxe est générée.

Visual Basic repère la plupart des erreurs de syntaxe dès que vous validez une ligne par Entrée. La Figure PB1.2 reproduit la boîte de dialogue qui s'affiche lorsqu'une ligne incorrecte a été saisie. (La boîte de dialogue ne parle pas toujours d'erreur de syntaxe ; le message d'erreur est parfois plus précis et désigne l'instruction fautive.) Si vous localisez le problème (dans ce cas, il s'agit d'un signe "égale" en trop), cliquez sur OK et corrigez. Si vous hésitez, cliquez sur le bouton Aide pour obtenir plus d'informations.

Visual Basic ne repère pas toujours avec justesse le problème dans une instruction fautive. Dans la Figure PB1.2, le problème est bien circonscrit ; mais il arrive que Visual Basic ne désigne pas le problème exact, peut-être parce qu'il ne s'en est rendu compte que quelques mots clés après. Auquel cas, vous aurez d'abord à vérifier vous-même les dernières lignes avant de remédier au problème.

Figure PB1.2
Visual Basic signale les erreurs de syntaxe dès que vous changez de ligne.



Les lignes 6 et 7 illustrent la première façon de documenter votre code. Comme nous l'avons vu au premier chapitre, la documentation est importante parce que vous aurez sûrement à revenir sur le programme par la suite ; et plus vous insérez de descriptions

détaillées dans le code, plus vite vous comprendrez ce qui y est écrit. Les lignes 6 et 7 sont des exemples de commentaires.

Visual Basic ignore complètement toute ligne qui commence par une apostrophe, signe indiquant que ce qui suit n'est pas du code. Rappelez-vous que les commentaires sont destinés aux humains, non aux ordinateurs.

Un commentaire est une note insérée dans le code et qui en décrit les sections. Un commentaire placé tout en haut du code permet aussi au programmeur de donner son nom et ses coordonnées. De cette façon, quiconque aura à modifier le programme pourra, au besoin, se référer à l'auteur.

Faire
Ajoutez en haut du code une ligne contenant la date et la description de toute modification apportée au programme. Cette sorte de journal de maintenance permettra, à vous ou à d'autres, d'établir un suivi précis des changements implémentés, et facilitera la maintenance.

Les commentaires, s'ils ne concernent pas directement Visual Basic, doivent accompagner les instructions. Il convient d'émailler le programme de descriptions en langage clair, qui permettent de comprendre immédiatement la fonction de telle ou telle partie du code. Comme le montre la ligne 11, un commentaire peut être placé sur la même ligne qu'une instruction, si du moins elle n'est pas trop longue. Essayez d'agencer le code et les commentaires de sorte que l'on n'ait pas à se servir des barres de défilement dans la fenêtre Code. La modification et le débogage du programme n'en seront que facilités.

Visual Basic dispose d'outils de "complément automatique". Par exemple, lorsqu'à la ligne 8 vous tapez `txtPassword.Text` *Visual Basic affiche les propriétés disponibles pour la zone de texte, et cela dès que vous avez tapé le point. Tapez* `T` *puis* `e`*, et Visual Basic vous amène directement à la propriété* `Text`*. Il suffit alors d'appuyer sur la barre d'espace pour que le nom* `Text` *apparaisse en entier dans le code. Evidemment, dans ce cas précis, vous n'économisez que deux caractères ; mais le gain de temps sera plus appréciable pour des propriétés telles que* `lblPrompt.Caption`*.*

Les lignes 12, 13 et 14 constituent, en fait, une seule instruction Visual Basic. L'instruction est longue à cause du chemin d'accès du fichier image spécifié. Vous pourriez aussi bien mettre le tout sur une seule ligne, mais cette ligne excéderait alors de loin la largeur de la fenêtre Code. Visual Basic vous permet de poursuivre une même ligne *logique* sur plusieurs lignes *physiques*. Pour ainsi segmenter une instruction, il suffit d'insérer une

espace suivie d'un underscore (_). Ce caractère, placé à la fin d'une ligne, indique à Visual Basic que la ligne suivante n'en est que la continuation.

Le chemin d'accès spécifié à la ligne 12 suppose que vous avez installé avec Visual Basic les fichiers graphiques des exemples. Si tel n'est pas le cas, le chemin d'accès ne conduira nulle part. Il vous faudra alors rechercher (à l'aide de l'outil Windows du même nom) le fichier Coins.wmf sur vos CD-ROM d'installation Visual Basic. Pour installer les images, insérez le premier CD-ROM Visual Basic, et sélectionnez Ajouter/Modifier une option, puis choisissez les fichiers correspondants.

Lorsque la ligne à fractionner est entre guillemets, la démarche est plus délicate. Vous devez placer des guillemets avant l'espace et l'underscore de bout de ligne, puis commencer la ligne suivante par une esperluette (&), suivie de nouveaux guillemets, et ainsi de suite pour toutes les lignes du commentaire. Vous comprendrez mieux le rôle de l'esperluette lorsque nous étudierons, au Chapitre 5, les chaînes de texte.

En avançant dans votre connaissance du langage de programmation Visual Basic, vous comprendrez pourquoi certaines instructions du Listing PB1.1 sont plus indentées que d'autres.

Un mot avant de terminer. Vous vous êtes peut-être demandé pourquoi l'utilisateur de notre application doit cliquer sur un bouton de commande pour "Tester le mot de passe". Pourquoi ne pas le laisser taper son mot de passe puis appuyer sur Entrée, au lieu de se servir de la souris ? Comme nous le verrons au Chapitre 7, la réaction à certaines touches, telle la touche Entrée, impliquerait du code supplémentaire. Cette application a été voulue aussi simple que possible. Car il vous en reste pas mal à apprendre…

Chapitre 4

Création de menus

La barre de menus offre un type spécial de contrôles qui permettent à l'utilisateur de sélectionner des options et d'exécuter des commandes. Une option de menu peut faire double emploi avec un contrôle de la feuille. Par exemple, la fenêtre d'application peut contenir un bouton Quitter, tout en proposant l'option de menu Fichier, Quitter. Dans les deux cas, la commande provoque la fermeture de l'application. Une option de menu reprend également la fonction d'un bouton de barre d'outils. Enfin, l'option de menu peut être le seul moyen d'exécuter une tâche précise. Avec Visual Basic, la création de menus est fort simple.

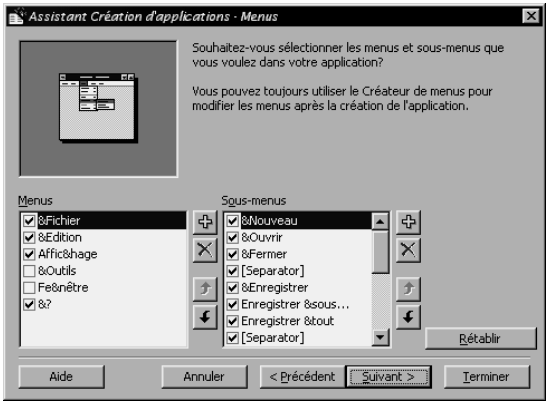
Voici ce que nous étudierons aujourd'hui :

- les menus générés par l'assistant Création d'applications ;
- les différents types d'options de menu qui vous sont proposés ;
- la création de barre de menus, de menus déroulants et de sous-menus ;
- les procédures événementielles de menus ;
- l'écriture rapide de procédures événementielles à l'aide des listes déroulantes Objet et Procédure de la fenêtre Code.

Créer des menus avec l'assistant Création d'applications

Vous avez pu voir, dans le cadre du premier chapitre, l'assistant Création d'applications en action. La Figure 4.1 montre l'écran de l'assistant Création d'applications depuis lequel vous pouvez créer des barres de menus.

Figure 4.1
L'assistant Création d'applications permet de générer facilement des barres et options de menu standards.



Une fois que vous avez fait votre choix parmi les options de menu et sous-menus proposés, l'assistant génère l'application et y intègre les contrôles correspondants.

L'avantage des menus, c'est que la plupart des utilisateurs savent les utiliser. Les utilisateurs manipulent spontanément les options de menu qu'ils connaissent.

Pour peu que vous ayez quelque expérience des applications Windows classiques, vous reconnaîtrez les options de menu proposées par l'assistant Création d'applications. Les menus tels que Fichier, Edition, Fenêtre, apparaissent dans de nombreux programmes Windows ; les options de menu disponibles, telles que Fichier, Quitter, sont tout aussi traditionnelles.

Même si vous ne fournirez sans doute jamais autant d'options de menu et de sous-menu que des applications de masse telles que Microsoft Word, il s'agit d'un bon modèle pour la conception de vos menus. A peu près tous les programmes Windows incluent une option Fichier, Quitter ; de même pour les options Edition, Copier ou Edition, Couper. Bien sûr vos applications répondent à des objectifs spécifiques, et n'ont donc pas à "coller" aux standards Windows. Toutefois, inspirez-vous de ces standards aussi souvent

que possible. Les utilisateurs n'en apprivoiseront que plus facilement vos applications. Et plus vite ils s'y habitueront, plus ils auront tendance à les utiliser — et plus vous aurez de clients fidèles.

L'assistant Création d'applications facilite considérablement la construction des menus. C'est pourquoi vous devriez aussi souvent que possible vous en servir pour les applications nécessitant un système de menus complexe. Les sections qui suivent montrent qu'il est facile de créer des menus à l'aide des outils Visual Basic. Mais c'est encore plus facile avec l'assistant Création d'applications : tout ce que vous avez à faire, c'est de cliquer sur l'option à inclure dans l'application finale. Même si vous ne prévoyez pas de barre d'outils, ni rien de ce que propose l'assistant Création d'applications, il vous permettra de partir d'une application aux menus tout faits, et pour lesquels il ne restera qu'à ajouter les détails et les options de menu spécifiques requis par votre projet.

Consultez, dans l'aide en ligne MSDN, le livre The Windows Interface Guide to Software Design. Le chapitre "Menus, controls, and toolbars" décrit les standards Windows en la matière. En suivant ces grandes lignes, vous vous assurez que votre application sera conforme, et aussi familière aux utilisateurs que possible.

Introduction aux menus

Les menus Windows ne vous sont assurément pas inconnus, ne serait-ce que ceux de Visual Basic, que vous avez eu à utiliser dans le cadre des chapitres précédents. La Figure 4.2 montre le menu Fichier de Visual Basic déroulé, et en détaille chacune des options. Même si vous êtes rompu aux programmes Windows, prenez le temps d'étudier les légendes de la figure ; elles indiquent la terminologie employée par Visual Basic pour désigner les différentes options. La suite de ce chapitre explique comment inclure ces éléments dans vos propres menus.

Si une option de menu de votre application doit afficher une boîte de dialogue, terminez le libellé de l'option par des points de suspension (voir Figure 4.2). Lorsque vous créez un menu à sous-menus, Visual Basic se charge d'ajouter la petite flèche noire.

Certaines options de menu donnent sur des menus additionnels. La Figure 4.3 montre le menu Affichage de Visual Basic, dans lequel l'option Barres d'outils affiche un sous-menu. Notez que, dans ce sous-menu, une option est cochée. Cela indique que l'utilisateur peut activer et désactiver l'option en la sélectionnant. Le Créateur de menus permet de créer des options de menu classiques ou cochables.

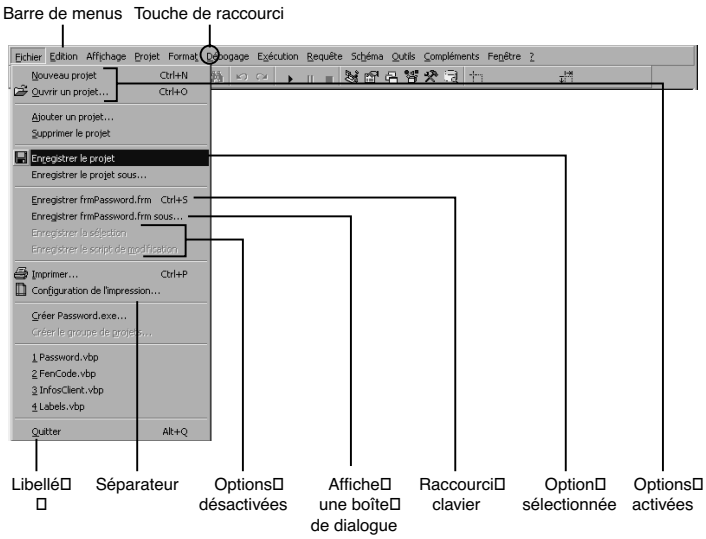


Figure 4.2
Visual Basic exploite les standards Windows en matière de menus et d'options de menu.

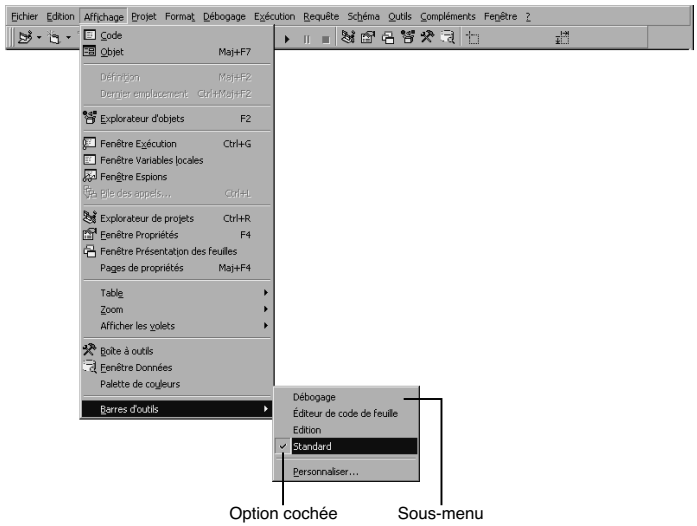
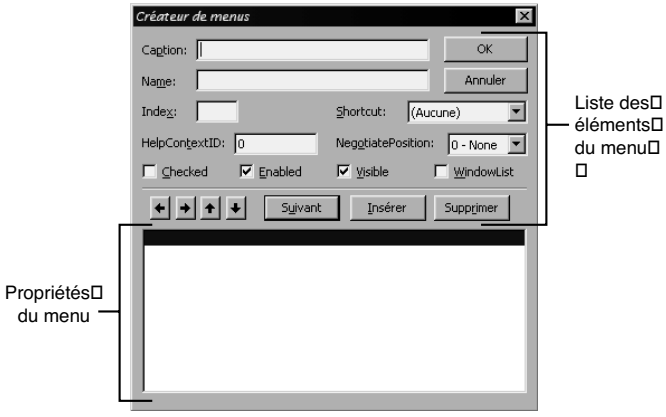


Figure 4.3
Visual Basic facilite la création de sous-menus.

Le Créateur de menus

La Boîte à outils ne contient aucun outil qui permet de créer des menus. Il faut, pour cela, recourir au Créateur de menus (voir Figure 4.4). Pour accéder à ce Créateur de menus depuis la fenêtre Feuilles, il suffit d'appuyer sur Ctrl-E (raccourci pour Outils, Créateur de menus).

Figure 4.4
Le Créateur de menus permet d'organiser les menus et sous-menus.



Le Créateur de menus facilite la création de menus pour vos applications. Le Créateur de menus fonctionne en quelque sorte comme une fenêtre Propriétés pour la barre de menus : vous y spécifiez le nom des contrôles ainsi que les libellés qui apparaîtront sur le menu, ainsi que d'autres informations corrélées. Le Créateur de menus est disponible dans les logiciels de programmation Microsoft depuis plusieurs années, et n'a que très peu changé depuis. Une telle longévité témoigne de la simplicité et de l'efficacité de cet outil.

La partie supérieure du Créateur de menus permet de spécifier les propriétés des contrôles correspondant à chaque option de menu. Les options peuvent apparaître directement sur la barre de menus, ou dans un sous-menu que l'on déroule. Dans la partie inférieure, la zone de liste affiche l'arborescence de votre système de menus à mesure que vous le construisez.

Fonctionnement du Créateur de menus

Le meilleur moyen d'apprendre à construire des menus à l'aide du Créateur de menus, c'est sans doute de créer une application simple qui contienne divers menus. Parce que vous ne maîtrisez pas encore les opérations sur fichiers et autres concepts avancés de

Visual Basic, cette application d'exemple ne sera pas tout à fait conforme aux standards Windows. Mais cela n'enlève rien à l'intérêt de l'exercice : reprendre les différentes étapes de la création de menus, et démontrer la simplicité d'emploi du Créateur de menus.

Voici les objectifs de l'application :

- afficher un label au centre de la feuille ;
- offrir une option de menu qui permette à l'utilisateur de changer la couleur de fond du label ;
- offrir une option de menu qui permette à l'utilisateur de changer le texte du label ;
- offrir une option de menu qui permette à l'utilisateur de quitter l'application ;

Ce programme, vous le voyez, sera fort simple, mais il n'en implique pas moins la création de menus fonctionnels.

Tout d'abord, ouvrez une nouvelle application EXE standard en choisissant Fichier, Nouveau projet. Assignez à la feuille le nom `frmMenu`, puis changez la propriété `Caption` pour qu'apparaisse sur la barre de titre `Application Menus`. Enfin, redimensionnez la feuille en donnant à `Height` la valeur `6030` et à `Width` la valeur `8415`.

Ajoutez à la feuille un label avec les propriétés suivantes :

Propriété	*Valeur*
`Name`	`lblMenu`
`Alignment`	`2-Center`
`BackColor`	`Red` (cliquez sur l'onglet Palette de `BackColor` et sélectionnez le premier rouge)
`BorderStyle`	`1-Fixed Single`
`Caption`	Sélectionnez une option de menu
`Font Size`	`14`
`Font Style`	`Bold`
`Height`	`615`
`Left`	`1800`
`Top`	`2160`
`Width`	`4935`

Info

Votre PC peut afficher de quelques centaines à plus d'un million de couleurs. A chaque couleur est associé un code numérique unique. Visual Basic utilise pour ces valeurs le système de numérotation hexadécimal. De votre côté, vous pouvez vous contenter de sélectionner les couleurs sur la palette, ce qui est quand même plus simple que de taper une valeur hexadécimale du genre `&H000000FF&`.

Info

Le système de numérotation hexadécimal, dit aussi base 16, est comme ces deux noms l'indiquent un système de numérotation en base 16. Dans le système de numérotation le plus courant, en base 10, il n'y a que dix chiffres : de 0 à 9. En base 16, il y en a seize : de 0 à 9, plus A, B, C, D, E et F. Les nombres hexadécimaux sont toujours précédés de `&H` *afin d'indiquer à Visual Basic (ainsi qu'au programmeur) qu'il ne s'agit pas d'une base décimale. Une même valeur numérique peut être exprimée en base 10 aussi bien qu'en base 16 ; mais l'hexadécimal, permet l'expression sous une forme plus compacte de valeur plus grande. Dans le cas qui nous occupe, plus d'un million de combinaisons de couleurs sont possibles. La base 16 permet d'exprimer chaque ton par un nombre de chiffres moindre que ne l'autoriserait la base 10.*

La Figure 4.5 montre ce à quoi votre écran devrait ressembler.

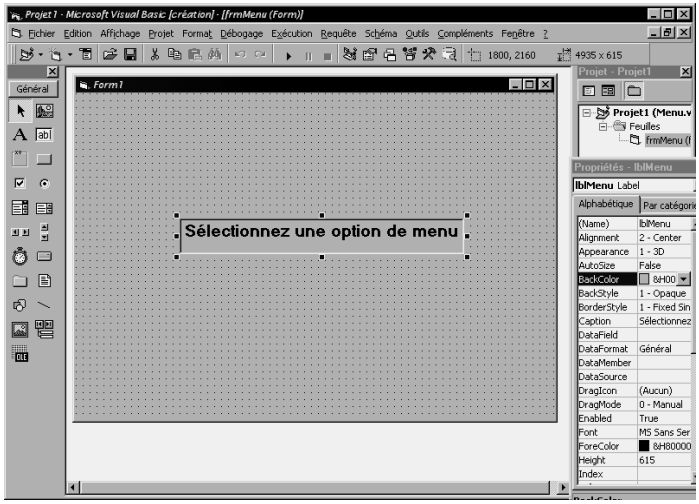


Figure 4.5
Le label a été placé ; on le contrôlera, par la suite, avec des options de menu.

Vous êtes maintenant prêt à créer le menu de l'application. Cliquez sur la feuille, puis faites Ctrl-E pour afficher le Créateur de menus. La plupart des *champs* du Créateur de menus sont vides ; ils attendent les valeurs que vous spécifierez pour chaque option.

Champs est un terme générique désignant l'emplacement dans lequel l'utilisateur ou le programmeur saisit des valeurs ; les champs prennent généralement la forme de zones de texte. Le Créateur de menus, comme la plupart des boîtes de dialogue, contient de nombreux champs.

Voici ce qui doit apparaître sur la barre de menus que nous allons maintenant créer :

- Fichier ;
- Couleur ;
- Message.

A ces menus devront être attribuées des touches de raccourci, de sorte que l'utilisateur puisse y accéder *via* le clavier. Les éléments que vous ajoutez dans le Créateur de menus peuvent apparaître directement sur la barre de menus, ou bien dans un menu déroulant, selon les valeurs que vous spécifiez. Suivez ces étapes pour ajouter le menu fichier à la barre de menus de l'application :

1. Tapez `&Fichier` comme contenu de `Caption`. Comme dans toutes les autres valeurs en Visual Basic, l'esperluette indique que le `F` pourra être utilisé comme touche de raccourci pour ce menu. A mesure que vous tapez le libellé, Visual Basic le reproduit dans la zone de liste en bas du Créateur de menus.
2. Appuyez sur Tab pour passer au champ `Name`. Tab et Maj-Tab permettent de passer le focus d'un champ à un autre.
3. Tapez `mnuFile` comme nom du contrôle.

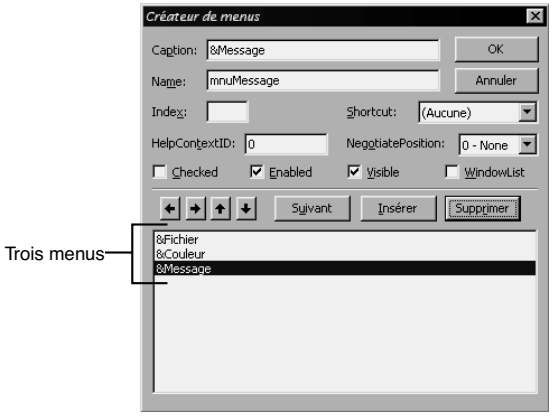
Gardez à l'esprit que les options de menu sont des contrôles qui, comme tous les contrôles, ont une propriété `Name`.

Faire

Vous devez adopter pour l'ensemble des menus une convention de dénomination cohérente. Apposez à tous les noms de menus le préfixe `mnu`, suivi du nom du menu ; ainsi, le nom du contrôle correspondant au menu Fichier serait `mnuFile` (ou, en code francisé, `mnuFichier`). Dans les noms des options de menu, reprenez le préfixe ainsi que le nom du menu ; ainsi, le nom du contrôle correspondant à l'option de menu Fichier, Quitter serait `mnuFileExit` (ou, en code francisé, `mnuFichierQuitter`).

4. Oubliez les autres champs et cliquez sur le bouton Suivant pour préparer le menu suivant (bon appétit). En cliquant sur le bouton Suivant, vous indiquez que la première option est complète.
5. Dans le champ `Caption` de l'option suivante, tapez `&Couleur` ; tapez `mnuColor` dans le champ `Name`.
6. Cliquez sur Suivant.
7. Tapez `&Message` comme `Caption` et `mnuMessage` comme `Name`. L'écran du Créateur de menus doit ressembler à la Figure 4.6.

Figure 4.6
Vous venez de créer les trois premières options de menu.



Tester le menu

Les menus peuvent être testés à tout moment durant la création. Cliquez sur OK pour fermer le Créateur de menus. La fenêtre Feuilles affiche la nouvelle barre de menus. Appuyez sur F5 pour exécuter l'application. Comme vous n'avez pas encore créé les sous-menus ni écrit les procédures événementielles, rien ne se passe lorsque vous cliquez sur les menus. Mais ces menus sont bien là, attendant la suite des opérations. Cliquez sur le bouton de fermeture de l'application en cours d'exécution (ou cliquez sur le bouton Fin de la barre d'outils Visual Basic), puis appuyez sur Ctrl-E pour revenir au Créateur de menus.

Ajouter un menu déroulant

Vous pouvez, au choix, créer les menus déroulants lors de la création de la barre d'outils, ou les ajouter plus tard. Toutefois, si vous créez d'abord la barre de menus complète (comme nous l'avons fait ici), il faudra insérer les options de menu à leurs emplacements respectifs dans l'arborescence. Le bouton Insérer du Créateur de menus est là pour ça.

Le bouton Insérer ajoute un menu ou une option de menu vierges. Avant de cliquer sur Insérer, sélectionnez l'option de menu située après l'emplacement de la nouvelle option. Par exemple, pour ajouter une option au menu Fichier, cliquez sur `&Couleur` *dans la zone de liste, puis cliquez sur Insérer. Visual Basic ajoute alors une ligne vierge entre* `&Fichier` *et* `&Couleur`*, dans laquelle la nouvelle option viendra prendre place.*

Visual Basic doit, d'une manière ou d'une autre, être en mesure de distinguer les menus comme tels d'avec les options de menu. Si vous insériez l'option Quitter après Fichier en utilisant la même technique que dans les sections précédentes, Visual Basic ajouterait cette option entre Fichier et Couleur, sur la barre de menus même, et non comme une *option* du menu Fichier.

Avant d'insérer une nouvelle option, cliquez sur le bouton figurant une flèche pointée vers la droite. Visual Basic ajoute des points de suspension après `&Fichier`, lesquels indiquent que l'élément en question est une option de menu, et non un menu. Dans le champ `Caption`, tapez `&Quitter`. Tout en tapant, vous voyez que le Créateur de menus insère l'option en bas de la zone de liste, indentée par rapport aux autres. Pour un sous-menu qui viendrait s'imbriquer dans un menu existant, il faudrait cliquer deux fois sur la flèche-droite, ce qui ajouterait deux séries de points de suspension.

Un trop grand nombre de niveaux de sous-menus deviendrait vite confus. Tenez-vous à un maximum de deux niveaux — un sous-menu accessible depuis un menu, puis éventuellement un autre sous-menu dans le sous-menu.

Nous avons utilisé, comme touche de raccourci de la commande Quitter, la lettre `Q`*. Pourquoi celle-là et pas une autre, outre qu'il s'agit de l'initiale ? Parce que c'est la touche de raccourci qu'utilisent les applications Windows standards, et que ces standards doivent être respectés autant que possible.*

Dans le champ `Name` de l'option, tapez `mnuFileExit`. Le menu Fichier et son contenu sont maintenant complets. Vous pouvez exécuter le programme pour vérifier le résultat.

Naturellement, rien ne se passe lorsque vous sélectionnez l'option Quitter, car la procédure événementielle requise n'est pas encore écrite.

Ajouter trois options à cocher

Le deuxième menu, Couleur, proposera trois options : Bleu, Vert et Rouge. Ces couleurs s'excluront mutuellement, le label ne pouvant être de plusieurs couleurs à la fois. Il s'agit donc de candidats idéaux pour des *options de menu cochables*.

Une options de menu cochable est une option de menu qui affiche une coche lorsqu'elle est active.

La Figure 4.7 montre le menu déroulant Couleur que nous allons maintenant créer. Comme vous pouvez le voir, l'une des options du menu est cochée, les deux autres ne le sont pas. En créant le menu Couleur, vous définirez l'option Rouge comme option active par défaut. Le label apparaîtra donc sur fond rouge jusqu'à ce que l'utilisateur sélectionne une autre couleur.

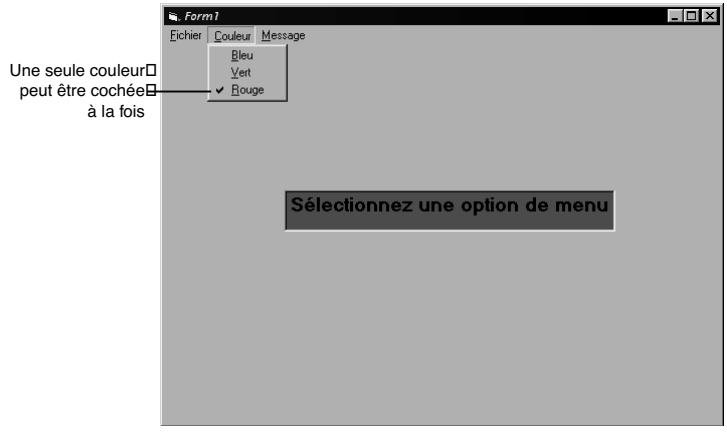


Figure 4.7
Seule l'option de menu cochée est active.

En fait, plusieurs options de menu peuvent être cochées à la fois. C'est par une programmation adéquate de la procédure événementielle que vous déterminez le caractère exclusif de la sélection. Ainsi, lorsque l'utilisateur cochera une option différente, l'option originale sera automatiquement désactivée.

Suivez ces étapes pour ajouter les options du menu Couleur :

1. Si vous l'aviez fermé, rouvrez le Créateur de menus.
2. Dans la zone de liste, cliquez sur l'option `&Message` pour la sélectionner.
3. Cliquez sur le bouton Insérer et sur la flèche-droite trois fois afin d'insérer trois lignes vierges pour les options.
4. Sélectionnez la première ligne vierge, où l'option Bleu va prendre place.
5. Tapez `&Bleu` dans le champ `Caption` et `mnuColorBlue` dans le champ `Name`. Lorsque l'utilisateur exécutera le programme pour la première fois, l'option Bleu ne sera pas cochée, indiquant par là qu'elle n'est pas sélectionnée.
6. Cliquez sur Suivant pour insérer la prochaine option.
7. Tapez `&Vert` dans le champ `Caption` et `mnuColorGreen` dans le champ `Name`.
8. Cliquez sur Suivant pour insérer la prochaine.
9. Tapez `&Rouge` dans le champ `Caption` et `mnuColorRed` dans le champ `Name`.
10. L'option Rouge devra être sélectionnée par défaut lorsque l'utilisateur lancera le programme pour la première fois. A cette fin, cochez la case `Checked` du Créateur de menus.
11. Fermez le Créateur de menus et lancez l'application pour tester le menu Couleur. La fenêtre devrait ressembler à la Figure 4.7.

Vous pouvez non seulement proposer des options de menu, mais également activer et désactiver ces options, dans le Créateur de menus, ainsi qu'à l'aide d'instructions Visual Basic. La Figure 4.8 montre le menu Débogage de Visual Basic avec plusieurs options activées et plusieurs autres désactivées. Les options désactivées s'affichent en gris, de sorte que l'utilisateur n'essaie même pas de les sélectionner.

En cochant la case Enabled du Créateur de menus, vous définissez dès la phase de création que l'option sera activée par défaut. Selon l'application, vous activerez certaines options dès qu'elles deviennent disponibles, et vous désactiverez celles que l'utilisateur n'est pas autorisé à sélectionner. Par exemple, la plupart des traitements de texte désactive l'option Coller du menu Edition jusqu'à ce que l'utilisateur copie quelque chose dans le Presse-papiers. Un menu d'application n'est, somme toute, qu'un ensemble de contrôles, dont les divers champs du Créateur de menus définissent les propriétés. Le code Visual Basic peut donc modifier la propriété `Enabled` de n'importe quel élément de menu pour lui donner la valeur `True` ou `False` (c'est ce que nous ferons au prochain chapitre), de sorte que l'option soit activée et disponible, ou bien désactivée et indisponible jusqu'à ce que l'utilisateur l'active de nouveau.

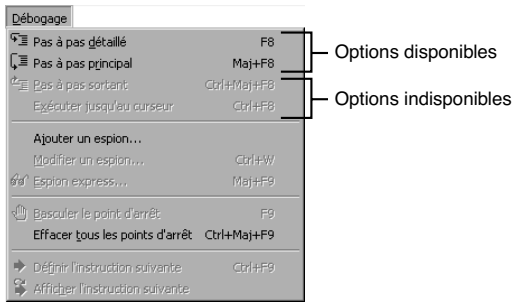


Figure 4.8
La propriété Enabled détermine quelles options seront disponibles ou indisponibles.

Notez que le menu Débogage, à l'instar de la plupart des autres menus, affiche les raccourcis clavier à droite des options de menu. F8 est le raccourci du mode Pas à pas détaillé, Maj-F8 celui du mode Pas à pas principal. (Dans le cas des combinaisons de touches qui lancent une option de menu, comme Ctrl-C pour Edition, Coller, on parle de *raccourcis clavier* plutôt que de touches de raccourci.) Pour affecter un raccourci clavier à vos options de menu, faites votre choix dans la liste déroulante Shortcut. Pour la plupart des raccourcis clavier, on utilise la touche Ctrl, comme dans Ctrl-C.

Vous pouvez remarquer que sept des options du menu Débogage (dont certaines sont désactivées) présentent à leur gauche une icône. Ces icônes renvoient au bouton correspondant de la barre d'outils. Ainsi, il existe sur la barre d'outils Débogage un bouton pour l'option Pas à pas détaillé. Les icônes permettent aux utilisateurs de repérer les boutons qui correspondent à telle ou telle option de menu. Malheureusement, Visual Basic ne donne aucun moyen d'adjoindre des icônes à vos propres options de menu (mais des contrôles externes sont disponibles à cet effet).

Ajouter le menu Message

Le menu Message proposera trois options, dont une seule pourra être sélectionnée à la fois (comme dans le menu Couleur). L'option cochée déterminera le message affiché dans le label. A titre d'exercice, vous allez formater le menu Message selon une manière différente de la méthode classique. Le menu Message devra ressembler à la Figure 4.9.

Notez dans le menu Message la barre séparatrice. Les barres séparatrices permettent de grouper les options. L'utilisateur ne peut sélectionner une barre séparatrice. Lorsque, dans un menu déroulé, l'utilisateur se déplace à l'aide de la touche flèche-bas, la sélection ne tient pas compte des barres séparatrices.

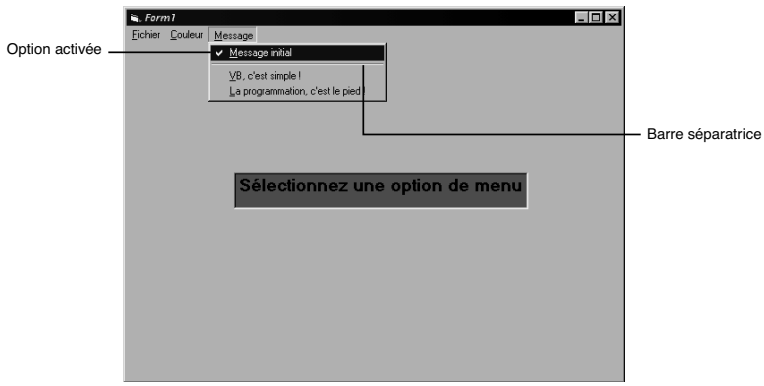


Figure 4.9
La barre séparatrice vous permet de grouper les options de menu.

Suivez ces étapes pour ajouter le menu Message et y inclure une barre séparatrice :

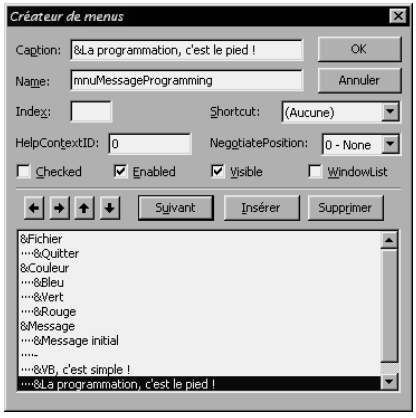
1. Appuyez sur Ctrl-E pour afficher le Créateur de menus.
2. Dans la liste déroulante, cliquez sur la ligne située en dessous de `&Message`, où l'option suivante va prendre place.
3. Cliquez sur la flèche-droite pour imbriquer l'option de menu dans le menu Message.
4. Tapez `&Message initial` dans le champ `Caption` et `mnuMessageInitial` dans le champ `Name`.
5. Cochez la case `Checked` pour que l'option soit sélectionnée par défaut au démarrage du programme.
6. Cliquez sur Suivant. Notez que Visual Basic décale automatiquement le nouvel élément suivant en fonction de l'indentation de l'élément précédent. (Pour supprimer cette indentation et faire de l'option un menu à part entière, cliquez sur la flèche-gauche ; pour faire remonter ou redescendre l'élément dans l'arborescence et l'imbriquer dans un autre menu ou sous-menu, cliquez sur la flèche-haut ou sur la flèche-bas.)
7. Dans le champ `Caption` de l'élément suivant, tapez un tiret simple (-). C'est le petit nom de la barre séparatrice. Lorsqu'un élément a pour `Caption` un tiret simple, Visual Basic le transforme en barre séparatrice.
8. Dans le champ Name de la barre séparatrice, tapez `mnuMessageSep1`. Les barres séparatrices sont des objets à part entière et, à ce titre, requièrent un nom unique. Les barres séparatrices suivantes du menu message pourront être nommées `mnuMessageSep2`, `mnuMessageSep3`, etc.

Les barres séparatrices ne peuvent être cochées ni désactivées, ni avoir de raccourcis clavier.

9. Cliquez sur Suivant.
10. Tapez `&VB, c'est simple !` dans le champ `Caption` et `mnuMessageSimple` dans le champ `Name`, puis cliquez sur suivant.
11. Pour l'option suivante, tapez `&La programmation, c'est le pied !` dans le champ `Caption` et `mnuMessageProgramming` dans le champ.

L'écran de votre Créateur de menus devrait ressembler à la Figure 4.10. Fermez-le, puis exécutez l'application pour tester le menu. Nous avons presque fini. Il ne reste qu'à ajouter le code qui rendra le menu opérant.

Figure 4.10
Votre menu est maintenant terminé.



L'assistant Création d'applications vous permet d'ajouter des barres séparatrices aux menus qu'il génère. Il suffit pour cela de sélectionner l'option `[Separator]`.

Finaliser le menu à l'aide du code

Pour que le menu que nous venons de créer soit opérant, il reste à lui affecter les procédures événementielles adéquates. Pas plus que dans les chapitres précédents, vous ne devez vous inquiéter des détails du code présenté ici. Il s'agit, pour l'instant, de bien comprendre les grandes lignes. Les menus et leurs options ne sont que des contrôles qui déclenchent des événements à l'exécution. Les utilisateurs interagiront avec votre application par l'intermédiaire des menus. Chaque fois qu'un utilisateur sélectionne un

menu, un événement `Click` est déclenché, et la procédure événementielle correspondante s'exécute.

Comme nous l'avons vu au chapitre précédent, le contrôle correspondant à l'option de menu `mnuFileExit` requiert, pour répondre à l'événement `Click`, une procédure événementielle `mnuFileExit_Click ()`. Et ainsi des autres options. Le Listing 4.1 fournit le code complet nécessaire à notre application. Ce code peut être entré de plusieurs manières :

- Vous pouvez écrire une procédure événementielle à la fois. Dans la fenêtre Feuilles, sélectionnez l'option de menu à laquelle vous souhaitez affecter le code. Visual Basic ouvre la fenêtre Code et insère automatiquement les *lignes d'encadrement* de la procédure. Il ne reste qu'à taper le corps. Lorsque vous en avez terminé, fermez la fenêtre Code, puis cliquez sur la prochaine option de menu à programmer ; et ainsi de suite.

Les lignes d'encadrement, "wrappers" en anglais, sont la première et la dernière ligne d'une procédure. Le chapitre précédent vous avait présenté le format de ces lignes d'encadrement, et avait expliqué leur nécessité.

- Après avoir entré la première procédure événementielle selon cette méthode, vous pouvez sélectionner l'option de menu suivante directement depuis la fenêtre Code. Vous pouvez voir, tout en haut de cette fenêtre, deux listes déroulantes : la liste Objet et la liste Procédure. La Figure 4.11 montre une liste Objet ouverte. Dans la liste Objet, sélectionnez la prochaine option de menu pour laquelle une procédure événementielle doit être écrite. Visual Basic insère les lignes d'encadrement de cette procédure juste en dessous de la précédente. Tapez le corps. Continuez ainsi jusqu'à ce que toutes les procédures événementielles des options de menu soient programmées.

Dans cette application, tous les objets, hormis le label, sont des contrôles de menu. Les contrôles de menu ne supportent que les événements `Click` ; aussi la liste Procédure n'affiche-t-elle que l'événement `Click`. Lorsque vous manipulez des contrôles supportant d'autres types d'événements, la liste Procédure est naturellement plus riche. Par exemple, si vous sélectionniez le contrôle du label dans la liste Objet, puis que vous ouvriez la liste Procédure, les noms d'événements proposés seraient bien plus nombreux, parce que le contrôle Label supporte plusieurs types d'événements.

- Vous pouvez enfin ouvrir la fenêtre Code et saisir le code du début à la fin. Cette méthode est un peu plus longue, car vous devez taper vous-même les lignes d'encadrement.

Figure 4.11
Les lignes d'encadrement de la nouvelle procédure peuvent être appelées depuis la fenêtre Code.

Les listes Objet et procédure de la fenêtre Code sont aussi utiles pour localiser du code que vous avez déjà entré, mais que vous souhaitez consulter ou modifier. Imaginons, par exemple, une application qui contient de multiples procédures événementielles pour plusieurs types de contrôles. Vous voulez atteindre la procédure `DblClick` *d'un bouton de commande spécifique. Sélectionnez, dans la liste Objet, le contrôle correspondant, puis choisissez l'événement* `DblClick` *dans la liste Procédure : Visual Basic affiche le code de cette procédure dans la fenêtre Code. Magique !*

Listing 4.1 : Le code du menu contrôle la couleur et le contenu du label

```
1:  Private Sub mnuColorBlue_Click()
2:  ' Colore le label en bleu et coche l'option de menu
3:  '    Bleu. S'assure que les options Vert
4:  '    et Rouge sont toutes les deux désactivées.
5:     lblMenu.BackColor = vbBlue
6:     mnuColorBlue.Checked = True
7:     mnuColorGreen.Checked = False
8:     mnuColorRed.Checked = False
9:  End Sub
10:
11: Private Sub mnuColorGreen_Click()
```

Listing 4.1 : Le code du menu contrôle la couleur et le contenu du label *(suite)*

```
12: ' Colore le label en vert et coche l'option de menu
13:  '    Vert. S'assure que les options Bleu
14:  '    et Rouge sont toutes les deux désactivées.
15:    lblMenu.BackColor = vbGreen
16:    mnuColorBlue.Checked = False
17:    mnuColorGreen.Checked = True
18:    mnuColorRed.Checked = False
19: End Sub
20:
21: Private Sub mnuColorRed_Click()
22: ' Colore le label en rouge et coche l'option de menu
23:  '    Rouge. S'assure que les options Bleu
24:  '    et Vert sont toutes les deux désactivées.
25:    lblMenu.BackColor = vbRed
26:    mnuColorBlue.Checked = Fasle
27:    mnuColorGreen.Checked = False
28:    mnuColorRed.Checked = True
29: End Sub
30:
31: Private Sub mnuFileExit_Click()
32: ' Termine le programme
33:    End
34: End Sub
35:
36: Private Sub mnuMessageInitial_Click()
37: ' Restaure le texte initial du label et
38: '    coche l'option de menu correspondante.
39: '    S'assure que les autres options sont désactivées.
40:    lblMenu.Caption = "Sélectionnez une option de menu"
41:    mnuMessageInitial.Checked = True
42:    mnuMessageProgramming.Checked = False
43:    mnuMessageSimple.Checked = False
44: End Sub
45:
46: Private Sub mnuMessageProgramming_Click()
47: ' Change le texte du label et coche
48: '    l'option de menu correspondante.
49: '    S'assure que les autres options sont désactivées.
50:    lblMenu.Caption = "La programmation, c'est le pied !"
51:    mnuMessageInitial.Checked = False
52:    mnuMessageProgramming.Checked = True
53:    mnuMessageSimple.Checked = False
54: End Sub
55:
56: Private Sub mnuMessageSimple_Click()
57: ' Change le texte du label et coche
58: '    l'option de menu correspondante.
59: '    S'assure que les autres options sont désactivées.
60:    lblMenu.Caption = "VB, c'est simple !"
```

```
61:    mnuMessageInitial.Checked = False
62:    mnuMessageProgramming.Checked = False
63:    mnuMessageSimple.Checked = True
64: End Sub
```

Encore une fois, ne vous souciez pas des détails du code. Mais assurez-vous de bien saisir le fonctionnement des procédures avant d'aller plus loin. A partir du chapitre suivant, vous commencerez à comprendre les subtilités du langage Visual Basic ; et une bonne compréhension des procédures événementielles vous sera alors indispensable.

Exécutez le programme et testez-le. Sélectionnez à plusieurs reprises les diverses options des menus Couleur et Message afin de vérifier que les coches suivent bien vos sélections et que le label reflète bien les changements. Vous pouvez changer la couleur et le texte dans n'importe quel ordre.

En résumé

Ce chapitre vous a présenté le fonctionnement des menus Visual Basic. Le Créateur de menus fonctionne un peu comme la fenêtre Propriétés, et vous permet de définir facilement les propriétés des contrôles du menu. Le Créateur de menus vous permet également de gérer les propriétés `Checked` et `Visible`, ainsi que d'attribuer des raccourcis clavier aux diverses options de menu.

Le chapitre suivant vous invite à plonger dans les profondeurs du langage de programmation Visual Basic. Vous découvrirez de quelle manière Visual Basic reconnaît et stocke les données, et apprendrez à définir les valeurs de propriétés lors de l'exécution à l'aide du code.

Questions-réponses

Q Pourquoi la barre de menus ne génère-t-elle pas d'événements `Click` ?

R En fait, c'est ce qu'elle fera si aucun contenu déroulant n'est défini pour le menu sélectionné. Si l'utilisateur sélectionne un menu qui ne contient ni options ni sous-menus, un événement `Click` est généré. Si, par contre, le menu se déroule pour afficher des options ou des sous-menus, cette action prend le pas sur l'événement `Click`.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Comment s'appelle la boîte de dialogue qui vous aide dans la création des menus ?
2. Les options de menu sont des contrôles. Vrai ou faux ?
3. Quel est l'événement que toutes les options de menu supportent ?
4. Dans quel contexte utilise-t-on l'expression *raccourci clavier* ?
5. A quoi servent les raccourcis clavier dans l'utilisation des menus ?
6. Quel événement les options de menu génèrent-elles si l'utilisateur y accède par un raccourci clavier ?
7. Le Créateur de menus vous aide à concevoir le système de menu et à créer les procédures événementielles `Click` des options de menu. Vrai ou faux ?
8. Quel est l'effet de la propriété `Checked` pour une option de menu ?
9. Plusieurs options de menu peuvent être cochées en même temps. Vrai ou faux ?
10. Dans le Listing 4.1, à quoi servent les lignes 57, 58 et 59 ?

Exercices

1. Expliquez la différence entre ces deux opérations du Créateur de menus : entrer un menu comme tel et entrer une option de menu.
2. **Chasse au bogue.** Manuel a des problèmes avec ses menus : les options restent cochées même lorsque l'utilisateur en sélectionne d'autres. Quel conseil donneriez-vous au pauvre Manuel ? (Vous n'avez pas à écrire de code.)
3. Attribuez des raccourcis clavier à chacune des options de menu créées aujourd'hui. Assurez-vous que ces raccourcis clavier soient bien tous uniques.

Chapitre 5

Analyse des données

Ici commence votre voyage au cœur du langage de programmation Visual Basic. Au lieu de manipuler des objets graphiques tels que les boutons de commande, vous allez maintenant taper les lignes de code qui feront de vos programmes des applications fonctionnelles et "intelligentes". Nous commencerons par étudier les données Visual Basic. Dans le chapitre suivant, vous apprendrez à travailler sur ces données à l'aide des commandes et des contrôles de la feuille.

Voici ce que nous étudierons aujourd'hui :

- l'utilisation avancée de la fenêtre Code ;
- les types de données qui peuvent être déclarés ;
- la distinction entre les types de données ;
- les conditions de stockage des données Visual Basic ;
- comment déclarer et assigner des valeurs aux variables ;
- l'ordre de préséance des opérateurs de calcul.

Notions préliminaires

Avant d'entrer dans le vif du code, quelques considérations préliminaires s'imposent. Vous devez comprendre de façon plus approfondie les rapports qu'entretiennent le code et les divers contrôles et feuilles d'une application. Tout d'abord, rappelez-vous que la fenêtre Projet permet d'afficher et de gérer l'ensemble des fichiers associés à votre application. Les procédures événementielles que vous écrivez dans la fenêtre Code ne nécessitent pas de fichiers propres. Elles sont stockées à même leurs contrôles respectifs. Les projets que nous avons étudiés jusqu'ici ne contenaient chaque fois qu'une seule feuille, laquelle incluait les divers contrôles et procédures événementielles correspondantes. Comme vous allez le voir, les feuilles ne contiennent pas que des procédures événementielles, mais également du code "général".

Nous avons vu au chapitre précédent que le code se présente sous la forme de procédures, et que Visual Basic supporte deux types de procédures : les sous-routines et les fonctions. Les procédures événementielles tombent dans la catégorie des sous-routines. Vous apprendrez bientôt à écrire des procédures de type fonctions. Quand une procédure n'est pas une procédure événementielle affectée à un contrôle particulier, le code prend la forme séparée d'un *module*. La fenêtre Projet affiche les modules du projet en cours (quand il en contient).

Un module est un fichier contenant du code de procédures. Le fichier qui contient la feuille et son code est un module de feuille. Vous avez donc, sans le savoir, déjà travaillé avec des modules.

Un projet qui contient plusieurs feuilles contient plusieurs modules de feuilles. En effet, les contrôles de chaque feuille devant répondre à des événements précis, chaque feuille a son propre code stocké dans un module de feuille. Lorsque vous incluez plusieurs feuilles à un projet, vous devez d'abord déterminer la feuille qui apparaîtra en premier au démarrage de l'application. La première feuille créée est la feuille de démarrage par défaut, mais vous pouvez en désigner une autre en sélectionnant, dans le menu Projet, la commande Propriétés de [*projet*], ou [*projet*] est le nom du projet en cours. Visual Basic affiche alors la boîte de dialogue Propriétés du projet (voir Figure 5.1). Vous apprendrez bientôt à programmer l'application de sorte qu'une feuille secondaire s'affiche au moment voulu.

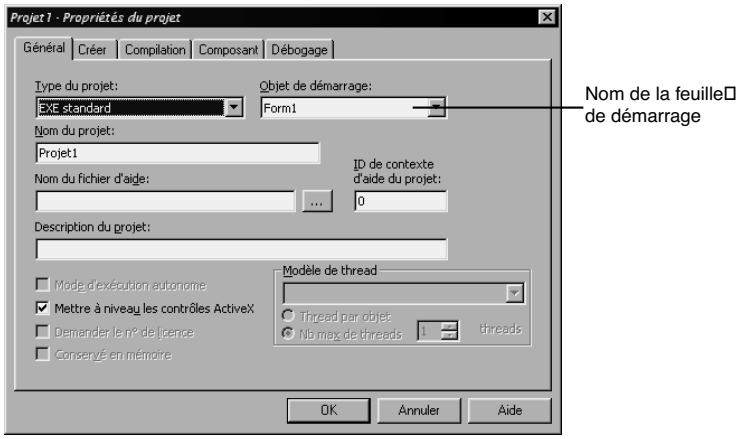


Figure 5.1
Dans la boîte de dialogue Propriétés du projet, vous spécifiez la feuille de démarrage.

Le contenu de la fenêtre Code

Les notions déjà acquises en matière de procédures événementielles vont maintenant vous servir à étudier les fondements du langage de programmation Visual Basic. Avant d'aller plus loin, toutefois, vous devez comprendre que les modules de feuilles ne contiennent pas seulement des procédures événementielles, mais aussi des *sections de déclarations*.

Les sections de déclarations réservent de l'espace et attribuent un nom aux zones de données utilisées dans le reste du module. Les contrôles n'ont pas à être déclarées dans ces sections. Mais vous aurez souvent, en revanche, à déclarer les autres zones de données Visual Basic.

Les sections de déclarations apparaissent toujours en haut de chacun des modules qui contiennent le code. Elles apparaissent donc avant toute procédure événementielle. Tout code entré avant la première procédure événementielle d'un module est considéré *général*, plutôt que lié à un contrôle spécifique. Pour commencer, vous déclarerez des données dans cette zone, traitant ainsi la zone entière comme une section de déclarations. Lorsque vous serez plus avancé, vous y écrirez également des procédure générales.

La fenêtre Code reproduite en Figure 5.2 devrait vous aider à mettre tout ça en perspective. Les détails ne sont pas importants pour l'instant ; seule compte l'idée générale. Dans le texte sélectionné, la section de déclarations s'ouvre sur l'instruction `Option Explicit`. Notez le contenu affiché des listes Objet et procédure de la fenêtre Code : `(General)` et

(Declarations). Grâce aux indications de ces deux listes, vous savez toujours dans quelle catégorie de programmation tombe une ligne de code.

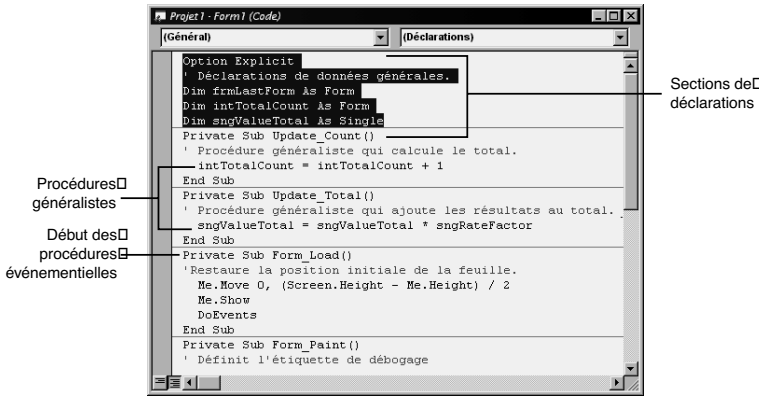


Figure 5.2
La fenêtre Code contient plusieurs sections.

Les deux procédures suivantes ne sont pas des procédures événementielles. Ce qui se voit à leur nom : rappelez-vous que le nom des procédures événementielles doit toujours contenir un underscore qui sépare le nom du contrôle de celui de l'événement. Si vous cliquiez dans la procédure `Update_Count()`, la liste Objet afficherait encore `(General)`, parce que le code fait partie de la section générale du module. La liste Procédure, en revanche, afficherait `Update_Count`, le nom de la procédure sélectionnée.

Voilà pour le tableau général. Nous entrons maintenant dans le vif du sujet.

Les données en Visual Basic

Les calculs peuvent impliquer différents types de données. Par exemple, vous pouvez travailler sur des noms, des adresses, des sommes d'argent, de grands et de petits nombres, et des données logiques du type vrai/faux, oui/non, etc. Afin de répondre à tous les besoins des différents programmes, Visual Basic supporte plusieurs types de données.

Comme la plupart des langages de programmation, Visual Basic est assez tatillon sur la question des données. Vous devez donc vous plier à ses exigences en la matière. Il est notamment obligatoire, avant de manipuler des données, d'en spécifier la nature exacte. Apprendre à programmer en Visual Basic, c'est avant tout s'initier aux différents types de données. Visual Basic en reconnaît douze.

Les données numériques

En règle générale, les données numériques relèvent d'une de ces deux catégories :

- **Entiers.** Nombres entiers sans décimale, tels que `614`, `0`, `-934` ou `3 918 938`. Les entiers peuvent exprimer un âge, un numéral, un nombre d'années, etc. En anglais, "entier" se dit *integer*.
- **Décimaux.** Nombres décimaux pouvant exprimer une valeur fractionnaire, tels que `8.709`, `0.005` ou `-402.35534`. Les décimaux peuvent exprimer une température, une somme d'argent, un taux d'intérêt, etc. Les décimaux exigent la virgule, même si la partie décimale, située après la virgule, est zéro.

Les valeurs numériques d'exemple sont appelées *littéraux*, ou parfois *constantes*, parce qu'ils ne changent jamais. La valeur 614 sera toujours 614. Dans une section ultérieure sur les variables, vous apprendrez à déclarer les valeurs changeantes.

Visual Basic stocke et traite différemment les entiers et les décimaux, même si la différence n'est pas toujours évidente pour nous autres humains. Par exemple, `-8` n'est pas la même chose, pour Visual Basic, que `-8.00`.

Certains types de données consomment plus de mémoire et sont moins efficaces, tandis que d'autres consomment moins de mémoire et sont plus rapides à calculer. On ne peut jamais juger "à l'œil" du poids en mémoire d'un nombre. Le nombre `29 999` occupe autant de mémoire que le nombre `701`.

En vous initiant aux types de données supportés par Visual Basic, vous apprendrez aussi combien de mémoire requiert chacun des types. Bien que les questions de mémoire ne soient pas aujourd'hui moins problématiques qu'auparavant, le programmeur souhaite toujours que son application s'exécute le plus rapidement possible. Si donc vous avez le choix entre plusieurs types de données pour une valeur, privilégiez le moins coûteux en mémoire.

Le Tableau 5.1 décrit les huit types de données numériques supportés par Visual Basic, le poids en mémoire de chacun, et la plage de valeurs couverte. C'est en fonction de ces deux derniers critères qu'il convient d'opérer ses choix. Si, par exemple, vous devez exprimer un nombre négatif, le type `Byte` est exclu. En revanche, pour exprimer des âges de personnes, le type `Byte` sera le plus approprié.

En anglais, byte signifie "octet", une unité de stockage en mémoire valant huit bits.

Certaines valeurs du Tableau 5.1 sont exprimées en notation scientifique.

La notation scientifique permet d'exprimer de façon abrégée (et approximative) des valeurs décimales très petites ou très grandes. Le `E` *signifie exposant. On utilise alors le type de données* `Single`*, pour "précision simple". La notation scientifique de haute précision emploie un* `D`*, pour "précision double" ; le type de données sera alors* `Double`*. Le* `E` *et le* `D` *peuvent aussi bien être écrits en minuscules. Pour obtenir l'équivalent numérique d'une valeur en notation scientifique, on multiplie le nombre situé à gauche du* `E` *ou du* `D` *par 10 à la puissance du nombre de droite. Si l'exposant est négatif (le* `E` *ou le* `D` *est suivi d'un signe moins), on divise le nombre situé à gauche du* `E` *ou du* `D` *par 10 à la puissance du nombre de droite. Par exemple,* `5.83E+5` *égale* 5.83×10^5*, soit* `5.83 x 100 000`*, soit une valeur* `Single` *de* `583 000`*. Dans ce cas précis, évidemment, on ne gagne pas grand-chose à utiliser la notation scientifique. Mais, dans le cas d'une valeur de* `5.83E+125` *(583 suivi de 123 zéros), l'avantage est évident. Le nombre* `-3.2D-6` *égale* `-3.2/1 000 000`*, soit* `.0000032` *dans une zone de stockage* `Double`*.*

Tableau 5.1 : Les huit types de données numériques supportés par Visual Basic

Type	*Stockage*	*Plage*
`Byte`	1 octet	De `0` à `255`
`Integer`	2 octets	De `-32 768` à `32 767`
`Long`	4 octets	De `-2 147 483 648` à `2 147 483 647`
`Single`	4 octets	De `-3.402823E+38` à `-1.401298E-45` pour les valeurs négatives ; de `1.401298E-45` à `3.402823E+38` pour les valeurs positives.
`Double`	8 octets	De `-1.79769313486232E+308` à `-4.94065645841247E-324` pour les valeurs négatives ; de `4.94065645841247E-324` à `1.79769313486232E+308` pour les valeurs positives.
`Currency`	8 octets	De `-922 337 203 685 477.5808` à `922 337 203 685 477.5807` (cette extrême précision garantit l'exactitude à deux décimales près des calculs comptables et financiers).
`Decimal`	12 octets	`+/-79 228 162 514 264 337 593 543 950 335` sans décimale ; `+/-7.9228162514264337593543950335` avec jusqu'à 28 décimales après la virgule. (Le type de données `Decimal` n'est pas encore complètement supporté par Visual Basic, mais il assure la compatibilité avec des versions ultérieures.)

Si la question des types de données vous semble obscure ou secondaire, c'est que nous n'avons pas encore étudié les zones de stockage des données ; ce sera l'objet de la section sur les variables. Lorsque vous tapez la valeur littérale `-8.3` dans un programme, il n'est pas nécessaire de spécifier que ce littéral est de type `Single`. Ce dont vous devez vous soucier, en revanche, c'est du type de zone mémoire utilisé pour stocker cette valeur. Vous ne pouvez stocker `-8.3` comme un entier et espérer que Visual Basic traite la valeur adéquatement.

Il peut être utile, quand vous utilisez un littéral, de spécifier le type de données correspondant. Imaginons, par exemple, que la valeur `-8.3` est incluse dans un calcul mathématique de haute précision, combinée à des valeurs `Double`. Si Visual Basic suppose que `-8.3` est de type `Single`, le résultat pourrait ne pas présenter autant de décimales que l'exige la précision souhaitée. La solution est d'adjoindre à la valeur un suffixe spécifique qui indique à Visual Basic de traiter le littéral selon son type propre. Ainsi, si vous tapez `-8.3#`, Visual Basic traitera dans le calcul cette valeur comme `Double`, et donnera au résultat la plus grande précision décimale possible.

Tableau 5.2 : Suffixes Visual Basic désignant les types des littéraux

Suffixe	*Type de données*
`&`	`Long`
`!`	`Single`
`#`	`Double`
`@`	Currency

Le `E` et le `D` de la notation scientifique réprésentent respectivement les types `Single` et `Double`. Il est donc inutile d'adjoindre des suffixes de types de données aux littéraux exprimés sous cette forme.

Enfin une bonne nouvelle. En dépit de la lourdeur théorique de cette section, vous n'avez pas vraiment à vous inquiéter des types de données lorsque vous travaillez sur des valeurs littérales. Si vous devez attribuer un nombre à la propriété d'un contrôle, allez-y franco. C'est seulement dans des cas spéciaux, tels que des calculs mathématiques ou scientifiques pour lesquels une très haute précision est requise, que vous aurez à vous soucier du type des littéraux employés.

Autres types de données

Les types de données non numériques sont plus faciles à comprendre que les types numériques, surtout si vous n'êtes pas spécialement fort en maths. Si le BASIC et ses avatars sont restés dans la course depuis tant d'années, malgré la prolifération de langages prétendument "plus puissants", c'est sans doute en raison de sa capacité à traiter efficacement le texte. Le BASIC, et donc aussi Visual Basic, laisse tous les autres langages loin derrière lui quand il s'agit de traiter les *chaînes* de texte.

Une chaîne est une série de caractères (une chaîne de zéro caractère est encore une chaîne). Si ces caractères peuvent être des chiffres, le contenu d'une chaîne ne peut jamais être utilisé dans un calcul. En revanche, il est possible de se servir des chaînes pour des noms, des adresses, des codes, des numéros de sécurité sociale, etc. En fait, on y met à peu près tout. Les types de données numériques, quant à eux, ne sont utiles qu'à des fins de calcul, ou dans les cas d'informations numériques strictes, telles que des sommes d'argent.

Outre les données de chaîne, Visual Basic supporte plusieurs autres types de données, tels que les dates, les variables *booléennes*, etc.

On appelle booléennes, d'après le nom du mathématicien George Boole, les variables qui ne prennent que deux valeurs, exclusives l'une de l'autre. Ces valeurs sont souvent représentées comme "vrai" et "faux", bien que l'on puisse aussi bien parler de "oui" et "non".

Tableau 5.3 : Types de données non numériques supportés par Visual Basic

Type de données	*Stockage*	*Plage*
`String` (longueur fixe)	Longueur de la chaîne	De 1 à environ 65 400 caractères
`String` (longueur variable)	Longueur + 10 octets	De 0 à 2 milliards de caractères
`Date`	8 octets	Du 1er janvier 100 au 31 décembre 9999
`Boolean`	2 octets	`True` ou `False`
`Object`	4 octets	Tout objet incorporé
`Variant` (numerique)	16 octets	Toute valeur aussi grande qu'une `Double`
`Variant` (texte)	Longueur + 22 octets	Comme les `String` à longueur variable

Les littéraux de chaînes doivent toujours être écrits entre guillemets. Les chaînes peuvent contenir n'importe quels caractères. Les littéraux suivants sont tous des chaînes valides :

```
"Hilare or de cymbale à des poings irrités"
"44-51-54-48"
"666, rue de la Bête"
""
```

Tout ce qui est entre guillemets est une chaîne, même la *chaîne nulle* de la dernière ligne.

Une chaîne nulle est une chaîne d'une longueur de zéro octet, dont on se sert parfois pour réinitialiser le contenu d'une chaîne (lui donner la valeur zéro). La valeur spéciale `Null` *représente ce type de chaînes. Visual Basic supporte aussi une valeur de chaîne spéciale dite chaîne vide, représentée par le mot clé* `Empty`*. Les chaînes vides sont semblables aux chaînes nulles, mais sont traitées d'une manière un peu différente : une propriété de contrôle qui contient le mot clé* `Empty` *sera interprétée comme n'ayant encore jamais été initialisée par quelque valeur que ce soit, pas même une chaîne nulle.*

La différence entre les chaînes de longueur fixe et les chaînes de longueur variable deviendra de plus en plus cruciale à mesure que vous approfondirez les méthodes Visual Basic de stockage des données.

Les littéraux, comme une date ou une heure, doivent être précédés de deux dièses (#). Visual Basic autorise à peu près tous les formats de date et d'heure. Ces formats dépendent des paramètres internationaux définis sur votre PC. Les littéraux suivants constituent tous des dates et des heures valides :

```
#4 juillet 1776#
#7:11 pm#
#19:11:22#
#1-2-2003#
#5-Déc-99#
```

Le type de données `Boolean` est approprié aux valeurs de propriétés qui ne peuvent recevoir que les valeurs mutuellement exclusives `True` et `False` ; par exemple, une valeur de propriété `Enabled`.

Le type de données `Variant` est employé pour toutes sortes de données, à l'exception des chaînes de longueur fixe. Le type `Variant` est utile à divers titres, notamment lorsqu'une zone de stockage doit recevoir des données de types différents. Une zone de stockage `Variant` peut ainsi servir de zone de stockage temporaire pour des données qui seront ultérieurement placées dans une zone de stockage plus spécifique.

Vous ne pouvez pas ne pas avoir entendu parler du fameux bogue de l'an 2000. Afin d'économiser de l'espace mémoire, les programmeurs ont, pendant de nombreuses années, enregistré les dates sous un format à deux chiffres. Pour les programmes conçus de cette façon, des problèmes sont susceptibles de se produire lorsque l'année à deux chiffres passera de 99 à... 00. Rassurez-vous, Visual Basic passera le cap de l'an 2000 ; son format de représentation interne des dates tient compte du millésime. Vos programmes Visual Basic ne devraient donc souffrir d'aucun dysfonctionnement à minuit, le 31 décembre 1999.

Néanmoins, certains programmeurs Visual Basic recourant à des bidouillages destinés à épargner du temps et de l'espace mémoire, certains codes peuvent cesser de fonctionner à cette date. En tant que nouveau venu dans le domaine de la programmation, gardez à l'esprit ce problème, et travaillez toujours sur des années à quatre chiffres.

Les variables

Les *variables* reçoivent différentes valeurs. Si ses valeurs peuvent changer, c'est que la variable n'est rien de plus qu'une zone de stockage, une sorte de boîte capable de contenir une seule valeur à la fois. Lorsque vous affectez à une variable une nouvelle valeur, la valeur précédente est remplacée. La valeur du littéral `54` ne changera jamais ; mais si vous stockez ce littéral dans une variable, la valeur `54` ne s'y maintiendra que tant que vous n'affectez pas une nouvelle valeur.

Une variable est une zone de stockage temporaire en mémoire, désignée par un nom unique. Les variables reçoivent les valeurs et les calculs intermédiaires attribués aux contrôles de la feuille et générés par eux.

Vous seul décidez du nom des variables de votre code. Deux variables différentes ne peuvent prendre le même nom dans une même procédure, car Visual Basic ne saurait alors les distinguer. A la différence des propriétés de contrôles, déjà nommées dans le langage, les variables ont pour seul nom celui que vous leur donnez. Avant de pouvoir utiliser une variable, il faut la déclarer, c'est-à-dire indiquer à Visual Basic son nom et le type de données qu'elle contiendra. Les variables ne peuvent contenir que le type de données pour lequel elles ont été déclarées. Ainsi, une variable déclarée comme `Byte` ne pourra pas contenir une valeur de chaîne. (Les variables déclarées comme `Variant` font toutefois exception à cette règle.)

Déclaration des variables

On déclare les variables avec l'instruction `Dim`, qui en définit le nom et le type de données. Pour chaque variable utilisée, il doit y avoir une instruction `Dim`. On peut, en fait, déroger à cette règle, mais il en résulte alors des programmes bâclés susceptibles de générer des erreurs. On affiche, *via* le menu Outils, Option de Visual Basic, la boîte de dialogue obtenue par la commande de menu Outils, Options de Visual Basic, dans l'onglet Editeur, l'option Déclaration des variables obligatoire engage Visual Basic à exiger la déclaration initiale. La section de déclarations du code inclut par défaut l'instruction suivante :

```
Option Explicit
```

Cette instruction indique à Visual Basic que la suite du code contient les déclarations de toutes les variables du module courant. Par exemple, si vous orthographiez mal un nom de variable dans le code, Visual Basic s'en avisera aussitôt. Si vous n'exigez pas la déclaration explicite des variables, Visual Basic traitera simplement les variables fautives comme non initialisées, et les calculs impliquant ces variables renverront des résultats erronés.

Si vous n'exigez pas la déclaration explicite des variables, Visual Basic considère toute variable non déclarée comme de type `Variant`.

Voici le format de l'instruction `Dim` :

```
Dim VarName As DataType
```

`VarName` est le nom assigné à la variable ; *`DataType`* est l'un des types de données présentés aux Tableaux 5.1 et 5.3. Si vous déclarez une variable à l'intérieur d'une procédure (qu'il s'agisse ou non d'une procédure événementielle), ce doit être tout de suite après la première ligne d'encadrement. La variable est alors disponible pour toute la procédure, et seulement pour cette procédure. Les autres procédures n'ont aucun accès à la variable, qui ainsi reste *locale*, c'est-à-dire valable uniquement pour la procédure qui la contient. Si, en revanche, vous déclarez une variable dans la section de déclarations du module, cette variable sera disponible pour toutes les procédures du module. La variable est alors *globale* pour le module. Toutefois, aucun autre module de l'application n'y aura accès. Il est possible de déclarer des variables globales pour un projet entier ; mais plus vos variables seront locales, moins vous risquez d'utiliser une même variable pour deux emplois différents.

Deux variables peuvent porter le même nom tout en étant bien distinctes, pour autant qu'elles soient déclarées localement dans des procédures différentes. En outre, deux procédures peuvent se partager une variable déclarée localement pour une seule des procédures. C'est au Chapitre 8 que vous apprendrez à partager les variables locales.

Si vous restez maître de vos variables, il convient néanmoins de respecter certaines règles de dénomination :

- Tous les noms de variables doivent commencer par une lettre de l'alphabet.
- Les noms doivent contenir des lettres ou des chiffres.
- Les noms ne doivent pas dépasser 255 caractères.
- Employez pour les noms un jeu limité de caractères spéciaux. Le plus sûr est de s'en tenir à l'underscore (_). En n'utilisant que des chiffres, des lettres et l'underscore, vous n'avez pas à vous soucier des caractères spéciaux non autorisés. Notamment, les noms de variables ne doivent pas contenir d'espaces.

Voici maintenant, en matière de dénomination, les conventions générales éprouvées et approuvées par la communauté des programmeurs :

- Incluez dans les noms de variables un préfixe désignant le type de données. Cela vous épargnera de consulter la section de déclarations du programme pour déterminer le type de données d'une variable, et ainsi vous risquerez moins de vous tromper. Le Tableau 5.4 présente les préfixes de noms de variables couramment utilisés.

On peut stocker des valeurs d'un certain type dans des variables déclarées pour un type différent, à condition que les deux types soient compatibles, notamment en taille. Par exemple, vous pouvez stocker une valeur de type `Byte` *dans une variable de type* `Integer`*, parce que ce dernier type couvre une plage de valeurs plus grande que le type* `Byte`*.*

- Employez des noms explicites, tels que `curTotal`, plutôt que des noms ambigus tels que `a` ou `curX1`. La documentation du code n'en sera que facilitée.
- Servez-vous des lettres capitales pour séparer les parties du nom. (Dans cet ouvrage, nous n'utiliserons pour les noms de variables que les lettres et les chiffres ; certains programmeurs préfèrent toutefois recourir à l'underscore, comme dans `curTotal_General`.)

Tableau 5.4 : Préfixes décrivant le type de données d'une variable

Préfixe	*Type de données*	*Exemple*
`bln`	`Boolean`	`blnBoutonEnabled`
`byt`	`Byte`	`bytLength`
`cur`	`Currency`	`curSales98`
`dte`	`Date`	`dteOverdue`
`dbl`	`Double`	`dblScientificAmt`
`int`	`Integer`	`intYear1998`
`lng`	`Long`	`lngWeatherDistance`
`obj`	`Object`	`objWorksheetAcct99`
`sng`	`Single`	`sngSales1stQte`
`str`	`String`	`strFirstName`
`vnt`	`Variant`	`vntValue`

Voici quelques déclarations de variables possibles :

```
Dim intTotal As Integer
Dim curSales99 As Currency
Dim dteFinal As Date
Dim strName As String
Dim blnIsChecked As Boolean
```

Le nom des variables de type `Boolean` doit évoquer une question à laquelle on répondrait par oui ou non (ou par vrai ou faux, `True` ou `False`), comme pour l'exemple `blnIsChecked` (`blnEstCoche` en français).

Vous pouvez également, en les séparant par des virgules, déclarer plusieurs variables dans une même déclaration `Dim` ; mais il faut alors inclure la clause `As` *`DataType`* pour chaque variable :

```
Dim intTotal As Integer, curSales99 As Currency
```

En l'absence de clause `As` *`DataType`*, Visual Basic déclare lui-même la variable comme `Variant`. Ainsi, ces deux instructions sont équivalentes :

```
Dim vntControlVal As Variant
Dim vntControlVal
```

Même si vous déclarez une variable `Variant`*, spécifiez toujours la clause* `As Variant` *afin de clarifier vos intentions*.

Déclaration des chaînes

La déclaration des variables est un peu subtile, car le type de données `String` est applicable à deux sortes de chaînes : longueur fixe et longueur variable. Les chaînes à longeur variable sont les plus courantes. Elles sont aussi plus faciles à déclarer en ce qu'elle suivent le même format de déclaration `Dim` que les autres. Voici deux exemples de déclarations de variables à longueur variable :

```
Dim strCityName As String
Dim strStateName As String
```

Les variables `strCityName` et `strStateName` peuvent toutes deux contenir des chaînes de n'importe quelle longueur. Si, dans `strCityName`, vous stockez d'abord `"Paris"`, et que y vous stockiez par la suite `"Iekaterinenbourg"`, la chaîne s'ajustera aux différentes valeurs. C'est sur des chaînes à longueur variable que vous travaillerez le plus souvent. Aussi ce livre ne s'attardera-t-il pas sur les chaînes à longueur fixe, à moins que la longueur ne soit importante, comme c'est notamment le cas avec les fichiers. Vous pouvez aussi limiter le nombre de caractères susceptibles d'apparaître sur un label, ou sur tout autre contrôle, en y assignant une chaîne de longueur fixe.

Les guillemets ne font pas partie de la chaîne, mais servent uniquement à délimiter le contenu littéral.

Voici le format des instructions `Dim` par lesquelles on déclare les chaînes à longueur fixe :

```
Dim VarName As String * Length
```

L'option `* Length` indique à Visual Basic que la chaîne déclarée ne devra jamais occuper plus de caractères que ne le spécifie `Length` (longueur). Voici la déclaration section de déclaration d'une variable dont on veut limiter la chaîne à un maximum de cinq caractères :

```
Dim strZipcode As String * 5
```

Si vous tentez de stocker plus de caractères que ne le permet une chaîne à longueur fixe, Visual Basic ne retiendra que le nombre spécifié de caractères, et le reste se perdra dans le néant numérique. De tels bogues sont souvent difficiles à retracer.

Stockage des données

Une fois la variable déclarée, vous pouvez y stocker des données. Les *instructions d'affectation* sont le moyen le plus simple de stocker les données dans des variables. Voici le format d'une instruction d'affectation :

```
ItemName = Expression
```

ItemName (nom de l'élément) peut désigner une variable déclarée (ce cera généralement le cas dans ce chapitre), mais également une valeur de propriété. *Expression* peut être :

- une expression mathématique ;
- un littéral ;
- une variable ;
- une expression logique ou de chaîne ;
- une valeur de propriété (les propriétés de contrôles sont de type `Variant`, mais Visual Basic les convertit dans un format précis lorsqu'elles sont stockées dans des variables) ;
- une expression mathématique, logique ou de chaîne, qui contient une combinaison de littéraux, de variables et de valeurs de propriétés.

Le concept d'expression peut vous sembler obscur à ce point ; disons qu'est expression tout ce qui peut devenir une valeur. Voici quelques exemples d'instructions d'affectation valides :

```
curSales = 5712.75
strFirstName = "Idulphe"
blnPassedTest = True
blnIsEnabled = lblTitle.Enabled
dblValue = 45.1#
intCount = intNumber
dteOld = #4-1-92#
sngOld97Total = sngNew98Total - 1000.00
```

La première instruction d'affectation est assez révélatrice. La valeur `5712.75` est stockée dans une variable nommée `curSales`. Un suffixe peut être ajouté après un littéral, comme dans la cinquième affectation, afin de spécifier le type de données pour les côtés de l'expression. Dans ce cas, en revanche, `45.1` est plus petit qu'un type `Double` ; si vous omettiez le suffixe, Visual Basic ferait automatiquement la conversion. Aux variables déclarées comme `Boolean` ne sont affectées que les valeurs `True` ou `False`, ou une valeur de propriété contenant `True` ou `False`. Notez, à la dernière affectation, la présence du signe moins. La prochaine section vous apprendra à écrire les expressions mathématiques.

Visual Basic supporte encore un ancien format d'instructions d'affectation qui commence par le mot clé `Let`*. Les instructions suivantes ont exactement le même effet :*

```
Let intCount = 1
intCount = 1
```

Dans les quatre premiers chapitres, vous avez appris à stocker des valeurs de propriétés à l'aide du code Visual Basic. C'est également ce à quoi servent les instructions d'affectation. Les instructions suivantes changent la valeur affichée par un label `lblTitle` :

```
lblTitle.Caption = "La tâche est terminée"
```

Tous les contrôles ont des propriétés par défaut ; c'est à ces propriétés par défaut que Visual Basic affectera automatiquement les valeurs pour lesquelles vous ne spécifez pas de propriété. La propriété par défaut d'un contrôle Label est `Caption` ; aussi l'affectation suivante est-elle équivalente à la précédente :

```
lblTitle = " La tâche est terminée "
```

Les affectations aux propriétés par défaut demandent moins d'écriture. Mais, encore une fois, plus votre code est explicite, plus la documentation sera facile, et plus il sera clair pour les lecteurs éventuels. Spécifiez toujours le nom de la propriété concerné par l'affectation, même s'il s'agit de la propriété par défaut. De cette façon, les instructions ne prêteront à aucune ambiguïté.

Dès qu'une instruction affecte une valeur à un contrôle, ce contrôle est mis à jour sur la feuille. Ainsi, si vous affectez une nouvelle valeur à un label, le label affiche automatiquement le nouveau contenu.

Les opérateurs Visual Basic

Visual Basic supporte de nombreux *opérateurs* mathématiques et de chaînes. Le Tableau 5.5 présente les plus courants. Ces opérateurs vous serviront dans les expressions impliquées dans des calculs, ainsi que pour le traitement des données.

Les opérateurs traitent les données en calculant ou en combinant des résultats. La plupart des opérateurs sont des symboles, tandis que d'autres, tel `Mod`*, ressemblent plus à des commandes Visual Basic.*

Tableau 5.5 : Opérateurs courants utilisés pour le calcul et le traitement des données

Opérateur	*Signification*	*Exemple*	*Résultat*
`^`	Puissance	`2 ^ 3`	`8`
`*`	Multiplication	`2 * 3`	`6`
`/`	Division	`6 / 2`	`3`
`+`	Addition	`2 + 3`	`5`
`-`	Soustraction	`6 - 3`	`3`
`Mod`	Modulo	`11 Mod 3`	`2`
`\`	Division entière	`11 \ 3`	`3`
`+` ou `&`	Concaténation de chaînes	`"Bon" & "jour"`	`"Bonjour"`

La puissance multiplie un nombre par lui-même autant de fois que le spécifie l'exposant. Ainsi, `2 ^ 3`, 2 à la puissance 3, est égal à 2 × 2 × 2, c'est-à-dire 8. Le calcul exponentiel peut être appliqué aux valeurs fractionnaires. L'exposant peut être négatif ; on obtient alors la n^e puissance du nombre. Le fonctionnement des opérateurs de multiplication et de division va de soi : l'expression `10 / 2` donne 5, tandis que `10 * 3` donne 30.

L'opérateur `Mod` renvoie le reste d'une division entière. Seules des valeurs de type `Integer` apparaissent de chaque côté de `Mod`. Si vous entrez des valeurs de types différents, Visual Basic la convertira et l'arrondira en nombre entier avant de traiter l'opération. Par exemple, `11 Mod 3` renvoie 2, puisque 11 divisé par 3 égale 3, reste 2. L'opérateur de division entière, `\` (notez qu'il s'agit de la barre oblique inverse, ou *backslash*, et non d'un slash), renvoie le quotient d'une division, ignorant le reste. Ainsi, `11 \ 3` donne 3, puisque 11 divisé par 3 égale 3, reste 2. (Avec l'opérateur `/`, 11 divisé par 3 donnerait une valeur fractionnaire du genre 3,666.)

L'opérateur de l'addition est un *opérateur surchargé*, c'est-à-dire capable effectuer deux opérations différentes selon les données qui l'entourent. Lorsque ce sont des chaînes que vous placez de chaque côté des signes + ou `&`, Visual Basic met bout à bout les deux chaînes, et traite le tout comme une seule et même chaîne ; cela s'appelle la *concaténation*. Visual Basic n'ajoute rien entre les chaînes concaténées ; si donc vous voulez une espace, il faut l'inclure comme troisième chaîne entre les deux autres.

Un opérateur surchargé est un opérateur qui permet des opérations différentes, selon le contexte.

La concaténation est la mise bout à bout de plusieurs chaînes en une même chaîne.

L'affectation suivante concatène les valeurs de deux labels en une seule chaîne variable, et place une espace au milieu.

```
strCompleteName = lblFirst.Caption & " " & lblLast.Caption
```

Afin d'éviter toute confusion avec l'opérateur d'addition lors de la maintenance du code, n'utilisez l'opérateur `&` que pour des concaténations de chaînes.

L'ordre des opérateurs

Visual Basic effectue les calculs selon un ordre strict prédéfini, illustré dans le Tableau 5.6. La puissance passe en premier, puis vient la multiplication, puis la division, et ensuite seulement l'addition et la soustraction, à moins que des parenthèses ne viennent modifier cet ordre.

Tableau 5.6 : Visual Basic respecte l'ordre des opérateurs dans les calculs

Priorité	*Opérateur*	*Exemple*	*Résultat*
1	Parenthèses `()`	`(2 + 3) * 7`	35
2	`^`	`2 ^ 3 + 1`	9
3	`*, /, \, Mod`	`2 + 3 * 7`	23
4	`+, -`	`10 - 4 * 2 + 1`	3

Sauf parenthèses, les résultats intermédiaires des multiplications ou des divisions d'une expression sont toujours calculés avant les additions ou les soustractions. La puissance a préséance sur toutes les autres opérations.

Si une même expression contient à la fois une multiplication et une division, Visual Basic les traite de gauche à droite, sauf s'il y a des parenthèses. Ainsi, l'expression `10 / 2 * 3` renvoie le résultat 15 : Visual Basic divise d'abord 10 par 2, ce qui donne 5, puis

multiplie par 3 ce résultat, ce qui donne 15. De même, dans une expression sans autres opérateurs ni parenthèses, les additions et les soustractions sont calculées de gauche à droite.

En cas de parenthèses imbriquées, Visual Basic traite d'abord les parenthèses intérieures. Ainsi, dans l'expression `(10 + 2 - (8 - 3)) + 1`, Visual Basic commence par calculer `(8 - 3)`.

En résumé

Ce chapitre vous a donné quelques notions préliminaires des subtilités du code Visual Basic. Vous avez d'abord étudié comment les sections de déclarations s'inscrivent dans le code général de l'application, avant d'approfondir votre connaissance du langage, notamment en ce qui concerne les types de données.

Comme nous l'avons vu, Visual Basic supporte plusieurs types de données. Vous devez non seulement apprendre à les distinguer, mais aussi à déclarer convenablement les divers types de données sur lesquels vous travaillerez. Les variables servent à stocker provisoirement des données lors de l'exécution d'un programme. Mais avant d'utiliser une variable, il faut la nommer et la déclarer. Les différents opérateurs Visual Basic vous permettent d'effectuer des calculs mathématiques, dont les résultats pourront être stockés dans des variables.

Le prochain chapitre vous emmène un peu plus loin dans les profondeurs de Visual Basic. Vous y découvrirez une nouvelle série d'opérateurs, qui permettent de comparer les données. Vous étudierez également les nouveaux contrôles et instructions qui exploitent ces opérateurs.

Questions-réponses

Q Pourquoi Visual Basic ne calcule-t-il pas tous les opérateurs de gauche à droite ?

R Visual Basic ne fait que se plier à l'antique hiérarchie des opérateurs algébriques. C'est la faute des mathématiciens ! Plus sérieusement, l'ordre des opérateurs permet d'éviter toute ambiguïté dans le code. En fait, vous n'avez pas forcément à vous soucier de cette hiérarchie : vous pouvez imposer votre propre ordre, en utilisant les parenthèses même là où elles ne sont pas nécessaires. Par exemple, l'affectation suivante donne exactement le même résultat avec ou sans les parenthèses, lesquelles ne sont là que pour ôter toute ambiguïté :

```
intValue = (8 * 9) + intResult
```

Q Pourquoi faut-il préférer les variables locales aux variables globales ?

R Vos progrès en programmation vous amèneront à mieux comprendre ce problème. La règle *de facto* veut que les variables locales soient toujours plus claires et plus sûres que les variables globales. Les procédures doivent êtres aussi épurées que possible. Elle ne doivent avoir accès qu'aux variables strictement nécessaires. Ce cloisonnement permet de se prémunir contre les bogues pernicieux, et souvent très difficles à localiser, que les variables globales sont susceptibles d'engendrer.

En revanche, comme vous l'avez sans doute déjà compris, les contrôles sont globaux pour tout le projet. Pour que les contrôles d'une feuille soient accessibles à tout le code d'un projet, il faut que chaque procédure puisse lire ou modifier les valeurs de propriétés de ces contrôles.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quel type de code contient la section de déclarations d'un programme ?
2. Comment faire en sorte qu'une procédure puisse accéder à une variable locale d'une autre procédure ?
3. La valeur d'un littéral ne change jamais. Vrai ou faux ?
4. La valeur d'une variable ne change jamais. Vrai ou faux ?
5. Pourquoi Visual Basic supporte-t-il deux types d'opérateurs de division ?
6. Qu'est-ce qu'un opérateur surchargé ?
7. Quel opérateur doit-on de préférence utiliser pour concaténer des chaînes ?
8. Quel type de données contient tous les autres types de données ?
9. Les préfixes sont obligatoires dans les noms de variables. Vrai ou faux ?
10. Quelles sont les deux manières de s'assurer que Visual Basic n'autorise aucune variable non déclarée ?

Exercices

1. D'après vous, que fera Visual Basic de la déclaration de variable suivante ?

```
Dim intCnt As Integer, abc, curSales As Currency
```

2. **Chasse au bogue :** Marie essaie de calculer une moyenne à l'aide de l'expression suivante. Pouvez-vous l'aider ?

```
sngAvg = sngGrade1 + sngGrade2 + sngGrade3 / 3
```

3. Quel est le résultat des formules suivantes ?

```
a. 1 + 2 * 4 / 2
b. (1 + 2) * 4 / 2
c. 1 + 2 * (4 / 2)
d. 9 \ 2 + 1
e. (1 + (10 - (2 + 2)))
```

4. Ecrivez des instructions d'affectation qui donnent l'équivalent Visual Basic des formules suivantes :

   a. $\dfrac{3 + 3}{4 + 4}$

      a = ...

   b. $x = (a - b) * (a - 2)^2$

   c. $\dfrac{a^{1/2}}{b^{1/2}}$

      f = ...

5. Le programme du premier Projet bonus sur les contrôles, propriétés et événements, incluait la procédure suivante :

```
1: Private Sub cmdTest_Click()
2: ' Cette procédure événementielle s'exécute dès que
3: ' l'utilisateur décide de tester le mot de passe saisi
4:    If txtPassword.Text = "SSM" Then
         ' Mot de passe correct
5:       Beep
6:       Beep       ' Afficher l'image
7:       imgPassword.Picture = LoadPicture("C:\Program Files\"
8:         & "Microsoft Visual Studio\Common\Graphics\MetaFile\"
9:         & "Business\coins.wmf")
       lblPrompt.Caption = "Aboule le fric !"
10:   Else
```

```
11:      lblPrompt.Caption = "Mot de passe incorrect -
         ➥Essayer encore "
12:      txtPassword.Text = ""    ' Effacer le mauvais mot de
         ➥passe
13:      txtPassword.SetFocus     ' Mettre le focus sur la zone
         ➥de texte
14:   End If
15: End Sub
```

Etudiez cette procédure pour voir les affectations à l'œuvre. Mais surtout, comprenez-vous maintenant pourquoi les longues instructions fragmentées en plusieurs lignes, telles que les lignes 7, 8 et 9, doivent inclure des esperluettes ?

PB2

Variables et expressions

Le code de ce Projet bonus met en œuvre des déclarations de variables, des instructions d'affectation et des expressions. Maintenant que vous êtes en mesure de concevoir une feuille et d'utiliser divers contrôles, vous devez vous attaquer aux subtilités du code et apprendre à activer les contrôles de l'application à l'aide d'instructions Visual Basic. Cela commence par le traitement des données.

Le Listing PB2.1 illustre les concepts étudiés au Chapitre 5. Afin que vous puissiez vous concentrer sur le code, aucune feuille visuelle n'y est décrite. Si toutefois vous voulez créer une feuille pour tester le code, il suffit de créer une feuille simple contenant trois labels, `lblGrossPay`, `lblTaxes` et `lblNetPay`, ainsi qu'un bouton de commande `cmdCalcPay` qui déclenchera le code. Votre feuille devrait ressembler à la Figure PB2.1. Après avoir effectué quelques calculs, le code affiche les résultats comptables dans les trois labels.

Listing PB2.1 : Ce code met en œuvre des variables et des instructions d'affectation

```
1:  Private Sub cmdCalcPay_Click()
2:  ' Calcule les trois variables de la paye.
3:     Dim intHoursWorked As Integer
4:     Dim sngRate As Single, sngTaxRate As Single
5:     Dim curTaxes As Currency, curGrossPay As Currency
6:     Dim curNetPay As Currency
7:
8:     ' Initialise les variables
9:     ' (En réalité, ces données viendraient de
```

Listing PB2.1 : Ce code met en œuvre des variables et des instructions d'affectation *(suite)*

```
10:    '  l'utilisateur ou d'un fichier).
11:    intHoursWorked = 40     ' Total des heures travaillées.
12:    sngRate = 7.8           ' Paye par heure.
13:    sngTaxRate = 0.42       ' Pourcentage de prélèvements.
14:
15:    ' Calcule les sommes
16:    curGrossPay = intHoursWorked * sngRate
17:    curTaxes = sngTaxRate * curGrossPay
18:    curNetPay = curGrossPay - curTaxes
19:
20:    ' Affiche les résultats dans les labels
21:    lblGrossPay.Caption = curGrossPay
22:    lblTaxes.Caption = curTaxes
23:    lblNetPay.Caption = curNetPay
24: End Sub
```

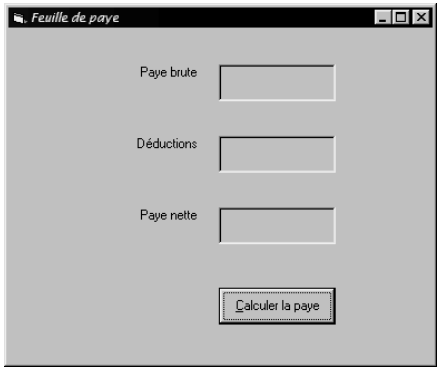


Figure PB2.1
Vous pouvez créer une feuille simple pour tester ce code.

Analyse

Les lignes 1 et 24 sont les lignes d'encadrement de la procédure événementielle du bouton de commande. Les lignes 2, 8, 15 et 20 sont des commentaires qui facilitent la documentation et la maintenance du code. Les lignes 3 à 6 déclarent les variables. Trois de ces variables sont de type `Currency`.

Les lignes 11, 12 et 13 affectent des valeurs aux trois variables. En situation réelle, ces données comptables seraient fournies par l'utilisateur, voire reprises d'un fichier. Mais vous n'avez pas encore étudié les entrées utilisateur ni la manipulation de fichiers ; du

reste, les instructions d'affectation se prêtent bien à notre exemple. Notez que, lorsqu'un littéral est affecté à une variable de type `Integer` (ligne 11), il n'y a pas de virgule décimale. En revanche, les valeurs assignées aux variables `Single` (lignes 12 et 13) contiennent de telles virgules.

Les lignes 16, 17 et 18 effectuent le calcul de la paye. Les expressions étant courtes, l'ordre des opérateurs n'a pas d'importance. Vous pouvez constater que ces expressions contiennent des types de données différents ; toutes ces données sont pourtant compatibles entre elles.

Enfin, les lignes 21, 22 et 23 affectent les valeurs de variables aux labels. Une fois les valeurs affectées, les contrôles de la feuille se mettent à jour et les labels affichent les résultats.

L'instruction d'affectation ne fait que "copier" le contenu de l'expression située à droite du signe = dans la variable ou le contrôle situé à gauche. Le contenu de l'expression n'est pas déplacé. Par exemple, une fois que la ligne 21 s'est exécutée, la variable `curGrossPay` *contient toujours sa valeur, mais cette valeur est également présente dans la propriété* `Caption` *du label.*

Chapitre 6

Opérateurs et instructions de contrôle

Ce chapitre traite encore des opérateurs Visual Basic, mais les opérateurs que vous allez étudier ne servent pas aux calculs mathématiques. Il s'agit d'opérateurs logiques et conditionnels destinés à comparer les données. Vous découvrirez également des instructions de contrôle grâce auxquelles vos programmes répéteront une section de code autant de fois que nécessaire, et interrogeront diverses conditions.

Voici ce que nous étudierons aujourd'hui :

- les opérateurs conditionnels qui permettent d'interroger les données ;
- les opérateurs logiques qui permettent de combiner les opérateurs conditionnels ;
- l'instruction `If` ;
- les boucles et leur utilité ;
- la différence entre les quatre types de boucles `Do` ;
- la différence entre les boucles `For` et `Do`.

Les opérateurs conditionnels

Imaginons que vous ayez à écrire une application comptable. Cette application doit totaliser l'ensemble des sommes dues à chaque fournisseur, puis imprimer le chèque. Mais qu'en est-il des fournisseurs avec lesquels la société n'a pas réalisé d'affaires depuis la dernière session de paiement ? Le programme doit-il imprimer un chèque de 0,00 F ? Evidemment non. Toutes les procédures que nous avons étudiées jusqu'ici se contentaient d'exécuter une instruction après l'autre. Grâce aux opérateurs conditionnels et aux instructions connexes que nous allons découvrir, vos programmes pourront modifier l'ordre d'exécution des instructions selon les données en présence. Ainsi, l'application comptable n'imprimerait de chèques que pour les fournisseurs auxquels vous devez de l'argent.

Le Tableau 6.1 présente ces nouveaux opérateurs Visual Basic. A la différence de ceux que nous avons étudiés au chapitre précédent, aucun de ces opérateurs n'effectuent d'opérations mathématiques. Il s'agit d'*opérateurs conditionnels* qui comparent les données. Grâce à eux, vos programmes Visual Basic seront plus intelligents. En comparant et en analysant les résultats, le programme peut décider par lui-même, et sur la seule base des données qui lui sont fournies, de ce qu'il convient de faire. En incluant à vos programmes opérateurs et instructions conditionnels, vous laissez Visual Basic décider, lors de l'exécution, quelles sont les instructions à exécuter.

Les opérateurs conditionnels permettent de comparer une valeur à une autre. Grâce aux opérateurs conditionnels, vous savez si une valeur est plus petite, plus grande qu'une autre ou lui est égale.

Tableau 6.1 : Visual Basic supporte six opérateurs conditionnels

Opérateur	*Description*	*Exemple*	*Résultat*
`=`	Egal à	`7 = 2`	`False`
`>`	Supérieur à	`6 > 3`	`True`
`<`	Inférieur à	`5 < 11`	`True`
`>=`	Supérieur ou égal à	`23 >= 23`	`True`
`<=`	Inférieur ou égal à	`4 <= 21`	`True`
`<>`	Différent de	`3 <> 3`	`False`

Remarquez, dans le Tableau 6.1, la colonne des résultats. Quel est le résultat de `6 > 3` ? Est-ce que 6 est plus grand que 3 ? Oui, et le résultat de l'expression conditionnelle est donc "vrai", `True`. Comme nous l'avons vu au chapitre précédent, Visual Basic supporte le type de données `Boolean`, qui n'accepte que les valeurs `True` ou `False`. Les mots clés `True` et `False` sont utilisés dans le code pour affecter des valeurs aux variables booléennes et aux propriétés de contrôles.

Nous avons vu que l'opérateur d'addition était surchargé. C'est également le cas de l'opérateur =. On l'utilise dans les instructions pour affecter des expressions aux variables et aux contrôles. On l'utilise aussi pour des comparaisons d'égalité. Visual Basic fait la distinction entre les deux en fonction du contexte dans lequel l'opérateur = apparaît.

Avant d'employer ces opérateurs dans votre code, assurez-vous d'en comprendre le fonctionnement. L'expression `23 >= 23` est `True` parce que 23 est supérieur *ou* égal à 23. Etudiez la colonne des résultats du Tableau 6.1 pour bien saisir ce concept.

Les littéraux ne sont pas les seules valeurs qui puissent apparaître de chaque côté d'un opérateur conditionnel. C'est aussi le cas des expressions, des variables, des contrôles et des combinaisons de tous ces éléments. Visual Basic traite de multiples types de données, et vos programmes doivent tester et comparer les données avant d'exécuter le code le plus approprié.

La présence d'une valeur `Null` d'un côté ou de l'autre d'un opérateur conditionnel constitue un cas particulier. Car Visual Basic ne renvoie alors, comme résultat de la condition, ni `True` ni `False`, mais... `Null`. Vous devez être attentif à ce que l'une des valeurs comparées puisse être `Null`. Dans ce cas, trois résultats sont possibles : `True`, `False` et `Null`. Parce que de tels résultats peuvent être déroutants, Visual Basic dispose d'un outil appelé *fonctions internes*, qui aide à détecter les valeurs `Null` ; vous étudierez cet outil au Chapitre 8, "Sous-routines et fonctions". Notez que les opérateurs conditionnels considèrent toute valeur `Empty` (correspondant à un contrôle ou à une variable qui n'a pas encore été initialisé par quelque valeur que ce soit) comme égale à zéro, ou comme une chaîne nulle si ce sont des chaînes que l'on compare.

Les opérateurs conditionnels comparent les chaînes exactement de la même manière que les valeurs numériques. Les comparaisons de chaînes suivent ces règles générales :

- Les lettres capitales valent moins que les minuscules ; ainsi, "BONJOUR" vient avant "bonjour".
- Les lettres se comparent selon l'ordre alphabétique ; ainsi, "A" vaut moins que "B", et le nom "Walter" passe avant le nom "William".
- Les nombres valent moins que les lettres ; ainsi, "3" est plus petit que "trois".

Si ces règles vous semblent obscures, rassurez-vous : Visual Basic compare la plupart des chaînes en suivant le même ordre que votre carnet d'adresses. Cette capacité à comparer les chaînes permet à vos programmes de classer les noms par ordre alphabétique, de tester les mots de passe, et d'analyser les données.

Une instruction spéciale peut apparaître dans la section de déclarations d'un module :

```
Option Compare Text
```

Cette instruction, éventuellement associée à l'instruction `Option Explicit`*, étudiée au chapitre précédent, a pour effet que les capitales et les minuscules sont égales en comparaison. Si vos comparaisons ne doivent pas tenir compte du style, incluez l'instruction* `Option Compare Text` *dans votre module. Cependant, un programme qui ne tient pas compte du style dans ses comparaisons ne donnera pas, dans la plupart des cas, un classement alphabétique fiable.*

Pour la comparaison des chaînes, Visual Basic suit l'ordre prescrit par la table ASCII — à moins que le module ne contienne l'instruction `Option Compare Text`*.*

La table ASCII (prononcer "aski") contient la liste de tous les caractères disponibles sur le PC, et attribue un numéro unique à chacun de ces caractères. La valeur ASCII de la lettre "A" est 65, celle de la lettre "B" 66, etc. Recherchez "ASCII" dans l'index de l'aide Visual Basic pour obtenir cette table (voir Figure 6.1). L'Annexe C de cet ouvrage reproduit également la table ASCII.

Voici quelques exemples de comparaisons de chaînes :

```
"abcdef" > "ABCDEF"
"Oui !" < "Oui ?"
"Bill vous regarde" = "Bill vous regarde"
"PC" <> "pc"
"Merci, Merci, Merci" >= "Merci, Merci"
```

Chacune de ces comparaisons de chaînes renvoie le résultat `True`.

Visual Basic supporte un autre opérateur conditionnel, `Like`, qui compare les valeurs sur la base de caractères génériques, ou "jokers". Vous avez sans doute déjà eu affaire aux jokers `*` et ? en travaillant sur des fichiers. Ainsi, `*` symbolise n'importe quel nombre de caractères (zéro inclus), tandis que ? symbolise un seul caractère. `Like` reconnaît un

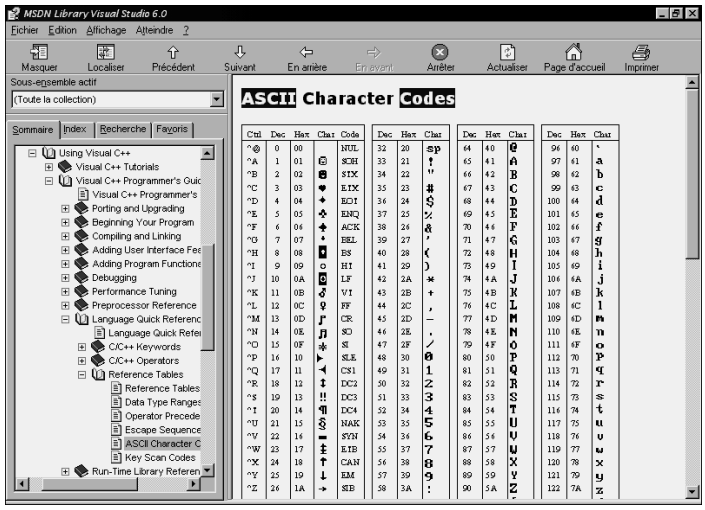


Figure 6.1
Pour la comparaison des chaînes, Visual Basic suit l'ordre des codes ASCII.

troisième joker, `#`, qui symbolise tout chiffre. Nous vous donnons ci-dessous quelques exemples d'utilisation de `Like`. Les expressions conditionnelles suivantes renvoient toutes le résultat `True` :

```
"Simon & Schuster Macmillan" Like "Si*"
"Qtr???" Like "QtrOne"
"Total##" Like "Total98"
"X" Like "[XYZ]"
```

Le dernier exemple relève d'un type spécial d'expression `Like`. Lorsque le caractère correspond à l'un des caractères entre crochets, le résultat renvoyé est `True`. Les crochets permettent d'interroger une série de caractères. Comparées à la chaîne `"Code[12345]Rouge"`, les valeurs suivantes sont toutes `True` : `"Code1Rouge"`, `"Code2Rouge"`, `"Code3Rouge"`, `"Code4Rouge"`, `"Code5Rouge"`.

En réalité, vos programmes Visual Basic compareront des variables et des contrôles dont le contenu est appelé à changer dans le cours de l'exécution. Les exemples ne comparent que des littéraux afin d'illustrer le fonctionnement des opérateurs.

Pour ces expressions, l'opérateur `=` aurait renvoyé la valeur `False`, car la condition d'égalité ne reconnaît pas les jokers.

Les données conditionnelles

Les valeurs comparées doivent toujours relever de types de données compatibles. Vous pouvez comparer entre eux des nombres de n'importe quels types de données. Vous pouvez comparez entre eux des chaînes ou des booléens. Mais vous ne devez jamais, par exemple, comparer une chaîne à un nombre, car le résultat risquerait fort d'être faux.

Les données de types `Boolean`, `Currency`, `String`, `Date`, et les différents entiers (`Byte`, `Integer` et `Long`) peuvent être comparés entre eux et répondre à une condition d'égalité. Ce n'est pas le cas des valeurs à précision simple ou double, comme dans `sngSales = sngGoal`. A cause de la façon dont Visual Basic stocke les données de précision, la comparaison de deux valeurs `Single` peut renvoyer un résultat d'inégalité ; en effet, Visual Basic arrondit de lui-même les valeurs, ce qui fausse les résultats. Pour tester l'égalité de deux variables de précision, il faut soustraire l'une de l'autre et mesurer la différence. De telles opérations s'avèrent assez fastidieuses, aussi vaut-il mieux les éviter.

Le type de données `Variant` se prête plutôt bien aux comparaisons conditionnelles. Vous aurez souvent à comparer la valeur d'un contrôle (une zone de texte, par exemple) à une variable ou à un littéral. Les propriétés de contrôles se comparent généralement comme des données `Variant`. Si un contrôle ou une variable `Variant` contiennent une valeur numérique, telle que `234.56`, et que vous compariez cette valeur à une variable numérique, Visual Basic procède en convertissant provisoirement la valeur `Variant` en nombre. Si, en revanche, vous comparez un contrôle ou une variable `Variant` à une chaîne, Visual Basic convertit provisoirement la valeur en chaîne, de sorte que la comparaison se fasse caractère par caractère, sur la base des codes ASCII. Ainsi, Visual Basic se charge des détails importuns qui, autrement, rendraient délicates les comparaisons impliquant un type `Variant`.

Si vous comparez une valeur numérique à une valeur `Variant`, et que cette dernière ne puisse être proprement convertie en nombre, Visual Basic générera une erreur d'exécution. Faites attention aux données utilisées. Les fonctions internes, dont nous parlerons au Chapitre 8, permettent d'interroger les types de données.

Combinaison d'opérateurs conditionnels et logiques

Techniquement, les six opérateurs conditionnels suffisent à interroger n'importe quelle condition. Mais on peut considérablement améliorer leur flexibilité en les combinant aux *opérateurs logiques* Visual Basic. Le Tableau 6.2 présente ces opérateurs.

Les opérateurs logiques permettent de combiner plusieurs séries de comparaisons conditionnelles. A l'instar de l'opérateur `Mod`*, les opérateurs logiques sont des mots clés, et non des symboles.*

Tableau 6.2 : Visual Basic supporte quatre opérateurs logiques

Opérateur	*Description*	*Exemple*	*Résultat*
`And`	Chaque terme de l'expression doit être `True`	`(2 < 3) And (4 < 5)`	`True`
`Or`	L'un des deux termes doit être `True`	`(2 < 3 ) Or (6 < 7)`	`True`
`Xor`	*Seul* l'un des termes doit être `True`	`(2 < 3) Xor (7 > 4)`	`False`
`Not`	Nie l'expression	`Not (3 = 3)`	`False`

Les opérateurs `And` et `Or` sont, de loin, les plus utilisés. L'opérateur `Xor` permet de distinguer deux options s'excluant mutuellement. Ainsi, dans une situation où une seule valeur doit être `True`, comme lorsque l'utilisateur doit sélectionner son mois de naissance, le résultat `False` d'une condition `Xor` indique que plus d'une option (ou aucune) a été sélectionnée. Enfin, l'opérateur `Not` nie une expression `True` ou `False`. Notez que cet opérateur doit être utilisé avec parcimonie, car il implique que l'on fasse abstraction de la logique en écrivant ou en déboguant le code.

L'expression suivante combine à des opérateurs conditionnels l'opérateur logique `And` :

```
(curSales < curMinSales) And (intYrsEmp > 10)
```

Si le chiffre de vente courant (`curSales`) est inférieur au minimum requis (`curMinSales`) et que le nombre d'années d'ancienneté (`intYrsEmp`) soit supérieur à 10, l'expression est entièrement `True` (et il y a du licenciement dans l'air !). Il serait tout à fait possible de tester chaque condition séparément, mais l'opérateur `And` permet de regrouper le tout en une même expression.

Ne combinez pas trop d'expressions conditionnelles avec des opérateurs logiques ; votre code deviendrait confus. Il convient de fragmenter les expressions trop complexes, telles que la suivante :

```
(a > 6) And (b < 1) Or Not(1 = c) Xor (d = 4)
```

L'ordre des opérateurs affecte le placement et l'exécution des opérateurs conditionnels et logiques. Considérez l'expression suivante :

```
curSales * sngCommission > curHighSales / 10
```

Quelles opérations Visual Basic traitera-t-il en premier ? Va-t-il d'abord comparer `sngCommission` à `curHighSales` pour ensuite multiplier le résultat par `curSales`, puis diviser ce dernier résultat par 10 ? Cela n'aurait aucun sens, puisque l'opérateur `>` ne peut renvoyer de résultats que `True` ou `False`, et qu'un tel résultat ne peut être impliqué dans un calcul mathématique.

Le Tableau 6.3 expose l'ordre des opérateurs de façon plus complète que le chapitre précédent. Il décrit le fonctionnement conjoint des opérateurs conditionnels et logiques contenus dans une même expression.

Tableau 6.3 : Ordre de préséance des opérateurs mathématiques, conditionnels et logiques

Priorité	*Opérateur*
1	Parenthèses
2	`^`
3	`*, /, \, Mod`
4	`+,-`
5	Opérateurs conditionnels tels que `Like`
6	`Not`
7	`And`
8	`Or`
9	`Xor`

Afin que vos programmes soient aussi clairs que possible, placez les expressions entre parenthèses ; vous éviterez ainsi toute ambiguïté dans l'ordre des opérations. Selon cette méthode, l'expression présentée plus haut ressemblerait à ceci :

```
(curSales * sngCommission) > (curHighSales / 10)
```

Les instructions *If*

`If` est l'une des commandes Visual Basic les plus utilisées. La commande `If` fait partie d'une instruction multiligne, l'*instruction `If`*, dont voici le format :

```
If condition Then
   Bloc d'instructions Visual Basic
End If
```

Ici, *`condition`* représente toute expression susceptible de renvoyer un résultat `True` ou `False`. Il peut s'agir d'une variable `Boolean`, d'un contrôle renvoyant une valeur `True` ou `False`, ou d'une expression plus longue incluant des opérateurs conditionnels et/ou logiques.

Visual Basic supporte toujours l'ancien format de `If`, hérité du BASIC, et qui s'écrit sur une seule ligne :

```
If condition Then instruction
```

Comme les instructions `If` donnent presque toujours sur plusieurs instructions, le format multiligne est plus approprié et plus répandu. Même dans le cas où le `If` ne déclenche qu'une seule instruction, le format multiligne est plus approprié, car des instructions supplémentaires pourront plus facilement être insérées.

Faire

Indentez le corps de l'instruction `If`, de sorte à repérer d'un simple coup d'œil son début et sa fin. Pour chaque instruction `If`, il existe, plus loin dans le programme, une instruction `End If`. Quelle que soit l'indentation du code, une instruction `End If` correspond toujours à l'instruction `If` la plus récente.

Sans s'en apercevoir, on emploie des instructions de type `If` tous les jours :

- Si (`If`) je reçois mon chèque, alors (`Then`) je payerai mon loyer dans les temps.
- Si (`If`) tu ranges ta chambre et (`And`) que tu finisses tes devoirs, alors (`Then`) tu sortiras jouer au foot.

Ainsi, les instructions Visual Basic `If` suivent un mode de raisonnement tout à fait commun. Elles fonctionnent de la manière suivante : le code contenu dans le corps de l'instruction ne s'exécute que *si et seulement si* la condition est remplie. Considérez de nouveau les raisonnements communs exposés ci-dessus. Si et seulement si vous touchez votre salaire, vous pourrez payer votre loyer à temps. Si vous ne recevez pas le chèque, eh bien... votre propriétaire attendra. Le second raisonnement implique que deux conditions soient satisfaites : si tu ranges ta chambre *et* si tu finis tes devoirs, alors seulement tu pourras aller taper dans le ballon.

Examinez l'instruction `If` du Listing 6.1 :

Listing 6.1 : Comparaison de données avec instruction If

```
1:  If (curSales > curSalesGoal) Then
2:     ' Ce commercial explose ses objectifs
3:     curSalaryBonus = 10000.00
4:     lblSalesNote.Caption = "Objectifs explosés !"
5:     lblSalesNote.BackColor = Red
6:     lblSalesNote.FontBold = True
7:  End If
8:  ' Le code continue ici
```

Si la valeur de `curSales` (chiffre de vente) est supérieure à la valeur de `curSalesGoal` (objectifs), les quatre instructions (sans compter le commentaire) des lignes 3 à 6 s'exécutent. Si la valeur est inférieure (ou même égale, le patron est exigeant), les lignes 3 à 6 ne s'exécutent pas. Dans tous les cas, le programme se poursuit à partir de la ligne 8, après que l'instruction `If` a fait ce qu'elle avait à faire. Ainsi, ce sont les données qui pilotent l'instruction `If`, et le programme prend une décision lors de l'exécution. Cette décision concerne l'exécution, ou non, d'une partie du code, à savoir le corps de l'instruction `If`.

Dans une instruction `If`, les parenthèses ne sont pas nécessaires autour de la condition ; en revanche, elles désignent clairement la condition à interroger, et en cela clarifient le code.

Les instructions *Else*

Nous venons de décrire un format d'instruction `If`. Mais les programmeurs emploient souvent un format plus étendu, qui ressemble à cela :

```
If condition Then
   Bloc d'instructions Visual Basic
Else
   Bloc d'instructions Visual Basic
End If
```

Comme pour toutes les instructions multilignes, l'indentation du corps est recommandée, mais non obligatoire. Selon le premier format d'instruction `If`, le code s'exécutait si et seulement si la condition était satisfaite ; mais rien n'indiquait au programme ce qu'il devait faire si la condition n'était pas satisfaite. L'instruction `Else` sert justement à cela. Une instruction `If... Else` contient deux corps distincts : l'un qui ne s'exécute que si la condition est `True`, l'autre qui ne s'exécute que si la condition est `False`. Quelle que soit la condition, le reste du programme se poursuit après que le test `If... Else` a été effectué.

Le programme du premier Projet bonus, "Contrôles, propriétés et événements", se servait d'une instruction `If... Else` pour tester la validité du mot de passe saisi. Cette partie du code est reproduite dans le Listing 6.2.

Listing 6.2 : Test du mot de passe par une instruction If

```
1:    If txtPassword.Text = "SSM" Then
2:       ' Mot de passe correct
3:       Beep
4:       Beep       ' Afficher l'image
5:       imgPassword.Picture = LoadPicture("C:\Program Files\"
6:         & "Microsoft Visual Studio\Common\Graphics\MetaFile\"
7:         & "Business\coins.wmf")
8:     lblPrompt.Caption = "Aboule le fric !"
9:    Else
10:      lblPrompt.Caption = "Mot de passe incorrect -
      ➥Essayer encore "
11:      txtPassword.Text = ""    ' Efface le mauvais mot de passe
12:      txtPassword.SetFocus     ' Met le focus sur la zone
      ➥de texte
23:   End If
```

La ligne 1 vérifie si la zone de texte contient le bon mot de passe. Si c'est le cas, le corps de l'instruction `If`, soit à partir de la ligne 2, s'exécute. L'instruction `Beep` provoque l'émission d'un bip par le haut-parleur du PC ; ainsi, si le mot de passe est correct, les lignes 3 et 4 s'exécutent, et l'ordinateur fait "bip bip". Les lignes 5, 6 et 7 affichent

l'image, tandis que la ligne 8 modifie le contenu du label. Au terme de l'instruction `If`, c'est-à-dire tout de suite après la ligne `End If`, le programme poursuit son exécution. Si toutefois le mot de passe saisi n'est pas le bon, et donc que la condition ne soit pas remplie, le corps de `Else` s'exécute (lignes 10, 11 et 12) pour indiquer à l'utilisateur que son mot de passe n'est pas valide.

Comme le montre le Listing 6.3, on peut imbriquer les instructions `If`.

Listing 6.3 : Les instructions If imbriquées permettent des comparaisons plus poussées

```
If (curSales > 100000.00) Then
  If (intHrsWorked > 40) Then
     curBonus = 7500.00
  Else
     curBonus = 5000.00
  End If
  lblBonus.Caption = "Bon boulot !"
End If
```

Dans les instructions ainsi imbriquées, chaque `Else` et chaque `End If` renvoie toujours au `If` le plus récent. Les différents niveaux d'indentation appliqués aux `If` imbriqués permettent de les circonscrire de façon claire.

Les instructions *Exit*

Il arrive parfois, selon les données, qu'une procédure (événementielle ou autre) doive être précipitamment interrompue. C'est ce que permet la combinaison des instructions `If` et `Exit`.

Voici le format de l'instruction `Exit` :

```
Exit Sub/Function/Do/For
```

Les slash indiquent que seul l'un des mots clés peut suivre l'instruction `Exit` ; tout dépend de ce dont on veut sortir. Pour sortir d'une procédure événementielle (c'est-à-dire d'une sous-routine, comme nous l'avons vu au Chapitre 4), l'instruction est `Exit Sub`. Pour sortir d'une fonction, ce serait `Exit Function`. Quant aux instructions `Exit Do` et `Exit For`, elles seront expliquées vers la fin de ce chapitre.

Le Listing 6.4 interrompt la procédure événementielle à la ligne 3 si la condition spécifiée dans l'instruction `If` est satisfaite.

Listing 6.4 : L'instruction Exit Sub permet d'interrompre une procédure

```
1:  Private Sub cmdCalc ()
2:     If (txtSales.Text < 50000.00) Then
3:       Exit Sub   ' Interrompt la procédure
4:     Else
5:       ' Si le chiffre de vente est au moins
6:       ' de 50 000 F, exécute l'instruction
7:       ' suivante, qui affiche le bonus comme
8:       ' pourcentage des ventes.
9:       lblBonus.Caption = txtSales.Text * .05
10:    End If
11: End Sub
```

Instructions *If... Else* imbriquées

Lorsque deux instructions `If... Else` sont imbriquées l'une dans l'autre, l'instruction intérieure doit utiliser le mot clé `ElseIf` à la place du simple `If`. Voyez le Listing 6.5.

Listing 6.5 : Le mot clé ElseIf permet de combiner les instructions If... Else

```
1:  If (intHours <= 40) Then
2:     curOverTime = 0.0
3:  ' Interroge les heures entre 40 et 50,
4:  ' et paye les 50 % d'heures sup.
5:  ElseIf (intHours <= 50) Then
6:        curOverTime = (intHours - 40) * 1.5 * sngRate
7:      Else
8:        ' Au-delà de 50, les heures doivent être payées
9:        ' doubles  ; entre 40 et 50 heures, la prime
10:       ' est de 50 %.
11:       curOverTime = ((intHours - 50) * 2 + (10 * 1.5))
        ➥* sngRate
12: End If
```

Une instruction imbriquée est une instruction qui apparaît à l'intérieur d'une autre.

L'instruction `ElseIf` de la ligne 5 commence un nouveau bloc `If... Else`. La ligne 1 vérifie si le nombre d'heures travaillées (`intHours`) est inférieur ou égal à 40 ; si tel n'est pas le cas, ce nombre doit logiquement être supérieur à 40. La ligne 5 vérifie si le nombre d'heures est inclus entre 40 et 50 (si le nombre est inférieur à 40, cette ligne ne s'exécute pas). La prime (`curOverTime`) de 50 % est alors calculée pour les heures supplémentaires. Si la condition de la ligne 5 n'est pas remplie, c'est que nécessairement le nombre

d'heures travaillées est supérieur à 50. L'expression de la ligne 11 calcule le paiement à 200 % des heures situées au-delà de 50, et le paiement à 150 % des heures situées entre 40 et 50.

Ces instructions `If... ElseIf... End If` imbriquées sont-elles particulièrement difficiles à déboguer ? Certes, et cet exemple simple en est la parfaite illustration. La section suivante décrit les instructions `Select Case`, qui offrent une solution plus intéressante.

Les instructions *Select Case*

L'instruction `Select Case` est la plus appropriée pour vérifier des conditions multiples. L'imbrication successive de trois ou quatre instructions `If... Else` complique considérablement le programme. On aboutit à une logique discutable du genre : "Si cela est vrai, alors si cela est vrai, alors si cela encore est vrai, alors faire cela ou cela, sinon..." L'instruction `Select Case` vous épargne ces imbroglios inutiles. En voici le format :

```
Select Case expression
  Case condition
     Bloc d'instructions Visual Basic
  [ Case condition1
     Bloc d'instructions Visual Basic]
  [ Case condition2
     Bloc d'instructions Visual Basic]
      :
  [ Case conditionN
     Bloc d'instructions Visual Basic]
  [Case Else
     Bloc d'instructions Visual Basic]
End Select
```

`Select Case` choisit entre plusieurs conditions. Le nombre de ces conditions, indiqué dans le corps `[ Case condition#...]`, varie selon la situation. Si aucune des conditions n'est remplie, le corps de `Case Else` (s'il y en a un) s'exécute.

Malgré un format un peu déroutant, `Select Case` est simple d'utilisation. Examinez le Listing 6.6.

Listing 6.6 : Les instructions Select Case comparent des valeurs multiples

```
1:  ' Interrogation d'une note scolaire
2:  Select Case txtGrade.Text
3:    Case "A"
4:       lblAnnounce.Caption = "Très bien"
5:    Case "B"
6:       lblAnnounce.Caption = "Bien"
```

```
7:    Case "C"
8:       lblAnnounce.Caption = "Peut mieux faire"
9:    Case "D"
10:      lblAnnounce.Caption = "Médiocre"
11:   Case "E"
12:      lblAnnounce.Caption = "Mauvais"
13:   Case Else
14:      lblAnnounce.Caption = "Note non validée"
15: End Select
```

Le type de données de `expression` doit être repris par chaque `condition` de `Case`. Le code du Listing 6.6 suppose que `txtGrade.Text` est une chaîne contenant une lettre : en effet, c'est bien à une chaîne que les lignes 3, 5, 7, 9 et 11 comparent la valeur de `txtGrade.Text`.

Si la zone de texte `txtGrade.Text` contient la lettre "A", le corps du `Case` de la ligne 3 s'exécute, et Visual Basic saute les autres conditions ; puis le code poursuit son exécution à partir de la ligne 15. Si `txtGrade.Text` contient la lettre "B", c'est le corps du `Case` de la ligne 5 qui s'exécute, et ainsi de suite. (Ce n'est pas le cas ici, mais le corps d'un `Case` peut courir sur plusieurs lignes.) Une fois qu'une condition `Case` est satisfaite, Visual Basic exécute tout le code compris jusqu'au prochain `Case` ; lorsque cela est achevé, l'instruction `Select Case` a fait son travail, et le programme peut continuer.

Si, pour une raison ou pour une autre, la zone de texte renvoie autre chose que A, B, C, D ou E, c'est le `Case Else` qui s'exécute et affiche dans le label un message d'erreur.

Visual Basic supporte un autre format de `Select Case`, qui permet de spécifier, par le mot clé `Is`, un opérateur conditionnel pour chaque *`condition`*. Le Listing 6.6 reprend notre exemple en mettant à profit ce format.

Listing 6.7 : Comparaisons conditionnelles dans Select Case

```
1:  ' Test d'une note scolaire
2:  Select Case txtGrade.Text
3:    Case Is >= 18
4:       lblAnnounce.Caption = "Très bien"
5:    Case Is >= 15
6:       lblAnnounce.Caption = "Bien"
7:    Case Is >= 12
8:       lblAnnounce.Caption = "Peut mieux faire"
9:    Case Is >= 10
10:      lblAnnounce.Caption = "Médiocre"
11:   Case Else
12:      lblAnnounce.Caption = "Mauvais"
13: End Select
```

Ici, chaque `Case` implique que la note se situe entre une valeur supérieure ou égale à 18 et une valeur inférieure ou égale à 10. Notez qu'aucun test spécifique n'est nécessaire pour les notes situées en-dessous de 10 car, en deçà de cette note, c'est le `Case Else` qui prend la relève. (Comme il ne s'agit que de montrer le fonctionnement de `Case Else`, cet exemple ne contient pas d'instruction de traitement d'erreur, et suppose donc que la note entrée se situe entre 0 et 20.)

Les instructions `Select Case` ne conviennent pas à tous les types de comparaisons. Les opérateurs d'inclusion ou logiques ne sont pas supportés ; les conditions `Case` ne peuvent donc pas être interrogées par `And`, `Or`, `Xor`, ni `Not`. Pour cela, la seule solution consiste à imbriquer des instructions `If... ElseIf... End If`.

Visual Basic supporte un troisième format de `Select Case`, dans lequel le mot clé `To` spécifie un ordre de choix. C'est cet ordre qui détermine quel corps de `Case` doit s'exécuter. Ce format est utilisé lorsque les valeurs testées sont séquentielles (voir Listing 6.8).

Listing 6.8 : Comparaisons de valeurs séquentielles dans Select Case

```
1:  ' Interrogation d'une note scolaire
2:  Select Case txtGrade.Text
3:    Case 0 To 9
4:       lblAnnounce.Caption = "Mauvais"
5:    Case 10 To 11
6:       lblAnnounce.Caption = "Médiocre"
7:    Case 12 To 14
8:       lblAnnounce.Caption = "Peut mieux faire"
9:    Case 15 To 17
10:      lblAnnounce.Caption = "Bien"
11:   Case Else
12:      lblAnnounce.Caption = "Très bien"
13: End Select
```

Vous pouvez remarquer que l'ordre des `Case` est inversé par rapport aux listings précédents, en raison du format utilisé. La première condition `Case`, ligne 3, teste la plage des notes les plus basses possible. Si la note renvoyée entre dans cette catégorie, le message "Mauvais" s'affiche dans le label. (Pour plus de simplicité, on présuppose que l'utilisateur entre des entiers. Ainsi, une note de `9.5` pourrait générer une erreur.) Les plages de valeurs se suivent séquentiellement. Vous pouvez également tester, de cette façon, des chaînes, en partant de la plus petite (en codes ASCII cumulés).

On peut combiner les différents formats de `Case` *à l'intérieur d'une même instruction* `Select Case`*. Voici un exemple :*

```
Case 101, 102, 201 To 205, Is > 300
```

Si l'expression spécifiée dans `Select Case` *renvoie une valeur égale à 101, 102, 201, 202, 203, 204, 205, ou supérieure à 300, le corps de* `Case` *s'exécute.*

Les boucles

Votre PC est puissant. Il peut traiter rapidement d'énormes quantités de données, telles que la comptabilité d'une grosse entreprise. Mais le traitement efficace de tant d'informations exige une technique spéciale : il s'agit d'intégrer le code dans des *boucles*, grâce auxquelles le programme analyse, et analyse encore, les données, jusqu'à ce que sortent les résultats attendus. Les *boucles* sont l'un des éléments les plus importants de la programmation.

Une boucle est une série d'instructions appelées à s'exécuter plusieurs fois. L'instruction de boucle se répète tant qu'une condition prédéfinie n'est pas satisfaite.

Les boucles nous renvoient au prochain chapitre, qui vous enseigne à recevoir de l'utilisateur des entrées (*input*) autrement que par les zones de texte — lesquelles ne conviennent pas à tous les types d'informations. Il arrive souvent qu'une question simple soit posée à l'utilisateur, qui répond en appuyant sur Entrée ; pour de telles réponses, les zones de texte ne sont pas ce qu'il y a de plus indiqué. (Les zones de texte sont parfaites pour les informations textuelles, telles que noms et adresses.)

Il est capital de bien comprendre les boucles avant d'aborder la question des entrées utilisateur : en effet, la réponse donnée n'est pas toujours celle qu'on attend. Imaginons que vous demandiez à l'utilisateur son âge, et qu'il réponde 291. Manifestement, il y a une erreur. Grâce aux instructions de boucle, la question peut être reposée jusqu'à ce que l'utilisateur donne une réponse raisonnable. Naturellement, le programme ne pourra jamais détermnier si l'utilisateur donne son âge réel ; mais vous pouvez faire en sorte qu'une réponse au moins plausible soit exigée. Les boucles peuvent répéter n'importe quel bloc de code.

Les boucles *Do*

La boucle `Do` est une instruction multiligne. Comme l'instruction `If`, l'instruction `Do` supporte différents formats :

```
Do While condition
   Bloc d'instructions Visual Basic
Loop
Do
   Bloc d'instructions Visual Basic
Loop While condition
Do Until condition
   Bloc d'instructions Visual Basic
Loop
Do
   Bloc d'instructions Visual Basic
Loop Until condition
```

Ici, *`condition`* peut être une expression, un contrôle ou une valeur `Boolean`. Le choix du format est, avant tout, une question de préférence et de style. Les points à considérer sont les suivants :

- **L'emplacement de la condition.** Si la condition apparaît au début de la boucle, dans l'instruction `Do`, le corps peut aussi bien ne jamais s'exécuter. Si, en revanche, la condition apparaît à la fin de la boucle, dans l'instruction `Loop`, le corps s'exécutera au moins une fois, puisque la condition n'est interrogée qu'en dernier.
- **La nature de la condition.** La boucle `Do` peut se répéter : a) *tant que* (`While`) la condition est remplie, b) *jusqu'à ce que* (`Until`) la condition soit remplie. Dans le premier cas, le corps cesse de s'exécuter dès que la condition n'est plus remplie. Dans le second, le corps cesse de s'exécuter dès que la condition est remplie.

La Figure 6.2 montre une boucle `Do` et illustre le processus de répétition. Ce code se contente d'augmenter une valeur (affichée comme `Caption` du label) par incréments de 1 ; lorsque la valeur de 10 est atteinte, la boucle s'arrête. (En réalité, un PC moderne exécuterait ce programme si rapidement que l'on n'aurait pas le temps de voir les valeurs s'afficher. Il s'agit d'un exemple.)

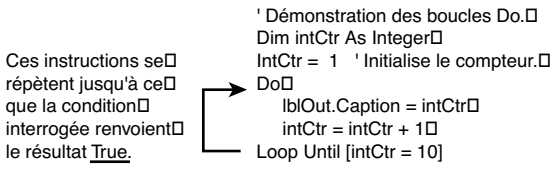


Figure 6.2
La boucle répète le corps d'instructions.

Le code de la Figure 6.2 illustre un type particulier d'affectation, dans lequel le même nom de variable apparaît de chaque côté de l'opérateur =. Une telle affectation ne sert, en fait, qu'à mettre à jour la valeur de la variable. Dans ce cas précis, l'instruction `intCtr = intCtr + 1` *ajoute 1 à la valeur de* `intCtr` *à chaque répétition de la boucle.*

Le corps du code de la Figure 6.2 s'exécute dix fois, et chaque fois la valeur de la variable `intCtr` est incrémentée de 1. Le format utilisé ici est `Do... Loop Until`, de sorte que la boucle se répète jusqu'à ce que `intCtr` soit égal à 10. Le Listing 6.9 présente une boucle semblable, mais selon le format `While... Loop`.

Listing 6.9 : L'instruction Do existe en plusieurs formats

```
1:  Do While intCtr <= 10
2:     ' Cette boucle fait la même chose
3:     ' que celle de la Figure 6.2
4:     lblOut.Caption = intCtr
5:     intCtr = intCtr + 1
6:  Loop
```

Vous devez, d'une manière ou d'une autre, modifier la condition contenue dans le corps de la boucle ; autrement, la boucle s'exécuterait indéfiniment. Si d'aventure vous écrivez une boucle sans fin, votre application se bloque jusqu'à ce que vous cliquiez sur le bouton Fin de Visual Basic ou sur le bouton de fermeture de l'application. Si rien, dans le corps de la boucle, ne permet de modifier la condition testée, la boucle continuera de s'exécuter.

Certaines boucles, notamment les boucles impliquant une entrée utilisateur, exigent que le corps s'exécute au moins une fois ; de là découle le choix du format de boucle. Si votre boucle doit s'exécuter au moins une fois, le format à utiliser est celui dans lequel la condition est interrogée à la fin de la boucle (voir Listing 6.10).

Les exemples de codes qui suivent ne sont pas complets. Les instructions censées traiter les entrées utilisateur et envoyer les messages d'erreur y sont remplacées par des commentaires. Concentrez-vous, pour l'instant, sur le fonctionnement des boucles. Le traitement des entrées utilisateur fera l'objet du prochain chapitre.

Listing 6.10 : L'utilisateur n'entre pas toujours des données valides du premier coup

```
1:  Dim strAns As String
2:  '
3:  ' L'utilisateur doit répondre par Oui ou Non.
4:  lblPrompt.Caption = "Continuer ? (Oui ou Non)"
5:  '
6:  ' Stockage de la réponse dans la
7:  ' variable chaîne nommée strAns.
8:  ' Test de la réponse et réitération
9:  ' de la question si nécessaire.
10: Do While (strAns <> "Oui" And strAns <> "Non")
11:   Beep    ' Avertissement
12:   lblError.Caption = "Merci de répondre par Oui ou Non"
13:   ' Stockage de la réponse dans la
14:   ' variable chaîne nommée strAns (rebelote).
15: Loop
16: ' Effacement du message d'erreur.
17: lblError.Caption = Null
```

La boucle `Do` commence à la ligne 10. Si l'utilisateur a entré `Oui` ou `Non` aux lignes 6 et 7 (rappelez-vous que les commentaires tiennent ici lieu d'instructions de traitement), la boucle affiche le message d'erreur de la ligne 12. Les commentaires des lignes 13 et 14 simulent une instruction qui recevrait de nouveau la réponse de l'utilisateur, et la ligne 15 renvoie la boucle à la ligne 10, qui teste de nouveau l'entrée. Combien de fois cette boucle s'exécutera-t-elle ? Soit jamais (si l'utilisateur a entré `Oui` ou `Non`), soit jusqu'à ce que l'utilisateur réponde correctement (non mais !).

Naturellement, la touche Verr Maj de l'utilisateur peut être ou non enclenchée ; et donc la réponse, être `OUI` ou `oui`, `NON` ou `non`. Dans ce cas, la ligne 10 échouerait, car les deux chaînes auraient des styles différents. Il vous faudra attendre le Chapitre 8 pour apprendre à tester les chaînes différentes.

L'instruction `Exit Do` permet d'interrompre une boucle avant son dénouement normal. Par exemple, vous traitez une série de factures dans une boucle qui se répète jusqu'à ce que le dernier numéro de compte client soit atteint. Si un mauvais numéro de compte est décelé par une instruction `If` à l'intérieur de la boucle, vous pouvez interrompre la répétion par `Exit Do` et afficher un message d'erreur.

Les boucles For

`For` est un autre type de boucle supporté par Visual Basic. Les boucles `For` exécutent une série d'instructions, un nombre prédéfini de fois ou jusqu'à ce qu'une condition soit satisfaite. Comme celui des boucles `Do`, le corps des instructions `For` est multiligne. En voici le format :

```
For intCounter = intStart To intEnd [Step intIncrement]
  Bloc d'instructions Visual Basic
Next [intCounter]
```

`intCounter` (compteur) est la variable numérique qui contrôle le corps de la boucle. A cette variable est affectée la valeur initiale *`intStart`*, avant la première *itération* de la boucle. La valeur *`intStart`* est généralement 1, mais peut être toute valeur numérique, variable ou valeur de contrôle que vous spécifierez. Chaque fois que le corps de la boucle se répète, la variable *`intCounter`* change de valeur (incrémentation ou décrémentation) en fonction de *`intIncrement`*. Si vous ne spécifiez pas de clause `Step`, l'instruction `For` utilise un incrément (ou *pas*) par défaut de 1. (La clause `Step` n'est donc pas obligatoire, et c'est pourquoi elle est indiquée entre crochets.)

On entend par itération chacun des cycles d'une boucle. Pour une boucle qui se répète trois fois, il y a donc trois itérations.

`intEnd` est un nombre, une variable ou une valeur de contrôle qui détermine la fin de la boucle. Lorsque *`intCounter`* est plus grand que *`intEnd`*, la boucle cesse et le code se poursuit à partir de la ligne suivant l'instruction `Next`. L'instruction `Next` referme le corps de la boucle, et relance l'itération. Si *`intCounter`* est plus petit que *`intEnd`*, Visual Basic augmente *`intCounter`* de la valeur *`intIncrement`*, et le corps de la boucle se répète de nouveau. (Notez qu'il n'est pas obligatoire de mentionner *`intCounter`* après `Next` ; cela ne sert qu'à rappeler de quelle boucle `For` spécifique l'instruction `Next` marque le terme.)

Quoi qu'il en soit, l'instruction `For` est très simple : elle ne fait rien d'autre que compter ou décompter, c'est-à-dire ajouter ou soustraire une valeur à une autre à chaque itération de la boucle. La boucle `For` incrémente si la valeur de `Step` est positive (elle l'est par défaut), et décrémente si la valeur de `Step` est négative.

Dans le Listing 6.11, la boucle `For` reprend la boucle `Do` du Listing 6.9. L'instruction `For` incrémente automatiquement la variable compteur du label.

Listing 6.11 : Les boucles For permettent d'incrémenter une variable compteur

```
1:  For intCtr = 1 to 10
2:    lblOut.Caption = intCtr
3:  Next
```

Quoi de plus simple ? Cette boucle se répète dix fois. La première fois que s'exécute la ligne 1, la valeur *intStart* (1 en l'occurrence, et valeur par défaut d'instruction `Step`) est affectée à `intCtr`. A la ligne 2, le corps de la boucle se sert de cette nouvelle valeur pour mettre à jour le label. La ligne 3 ordonne à la boucle de se répéter pour de nouveau incrémenter `intCtr` de 1, et cela jusqu'à ce que `intCtr` atteigne la valeur *intEnd* : 10.

L'instruction suivante est équivalente à celle de la ligne 3 du Listing 6.11, puisque la variable n'est qu'optionnelle dans l'instruction `Next` *:*

```
Next intCtr
```

Définir la valeur de Step

Dans le Listing 6.12, la boucle `For` commence à 10 et *incrémente* la variable de 5 jusqu'à ce que la valeur 100 soit atteinte.

Listing 6.12 : Boucle For avec valeur de Step positive (incrémentation)

```
1:  For intCtr = 10 to 100 Step 5
2:    lblOut.Caption = intCtr
3:  Next
```

Dans le Listing 6.13, la boucle `For` commence à 1000 et *décrémente* la variable de 100 jusqu'à ce que la valeur 0 soit atteinte.

Listing 6.13 : Boucle For avec valeur de Step négative (décrémentation For... Next;)

```
1:  For intCtr = 1000 to 0 Step -100
2:    lblOut.Caption = intCtr
3:  Next
```

Ces quelques courts exemples démontrent bien le comportement des valeurs *intStart*, *intEnd* et *intIncrement* dans la boucle. (Si la valeur de `Step` est négative, *intStart* doit être plus grand que `intEnd` ; autrement, le corps de la boucle ne s'exécuterait jamais.)

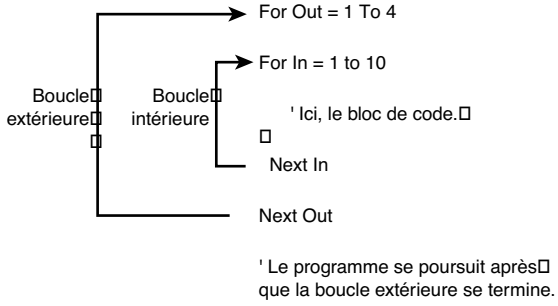
Ne confondez pas les boucles avec les instructions `If`*. Toutes reposent sur des valeurs conditionnelles ; mais, tandis que les boucles répètent leur corps autant de fois que nécessaire, le corps des instructions* `If` *ne s'exécute qu'une fois.*

L'instruction `Exit For` *permet d'interrompre une boucle avant son dénouement normal.*

Boucles For imbriquées

Comme toutes les autres instructions Visual Basic, les boucles `For` peuvent être imbriquées. Cela permet de répéter plusieurs fois l'ensemble d'une boucle. La Figure 6.3 donne le schéma d'une boucle `For` imbriquée. On pourrait dire, ici, que la boucle intérieure est plus "rapide" que la boucle extérieure. En effet, la boucle intérieure incrémente la variable `In` de 1 à 10 avant que la boucle extérieure n'ait complété sa première itération. La boucle extérieure ne se répète qu'à partir de l'instruction `Next Out`, qui n'est pas atteinte dans le code tant que la boucle intérieure n'a pas terminé son cycle. Lorsque l'instruction `Next Out` est atteinte, et que la boucle extérieure se répète, la boucle intérieure démarre de nouveau, et ainsi de suite jusqu'à la quatrième itération de la boucle extérieure.

Figure 6.3
La boucle extérieure détermine le nombre de fois que la boucle intérieure se répète.



En tout, la boucle intérieure de la Figure 6.3 s'exécute quarante fois. La boucle extérieure accomplit quatre itération, et à chacune de ces itérations la boucle intérieure se répète dix fois.

La Figure 6.4 montre deux boucles intérieures à l'intérieur d'une troisième. Les deux boucles intérieures doivent accomplir tout leur cycle avant que la boucle extérieure ne puisse terminer sa première itération. Lorsque la boucle extérieure attaque sa deuxième itération, les deux boucles intérieures se répètent de nouveau, et ainsi de suite.

Dans la Figure 6.4, le corps de chacune des deux boucles intérieures s'exécute quarante fois. La boucle extérieure accomplit quatre itérations, et chaque itération exécute d'abord la première, ensuite la seconde boucle intérieure ; puis la boucle extérieure repart, etc.

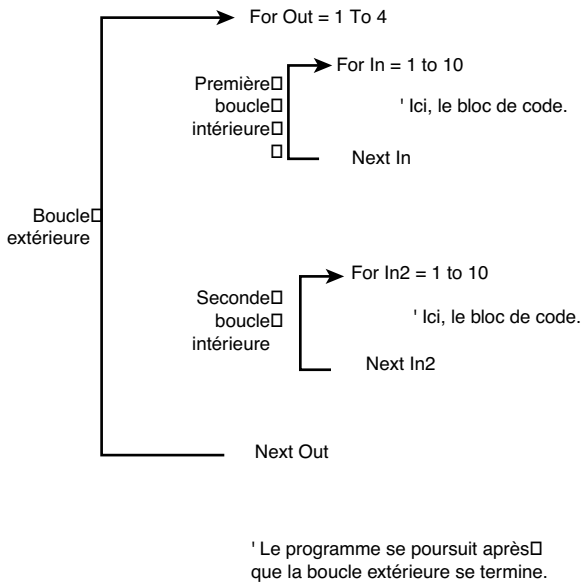


Figure 6.4
On peut imbriquer plusieurs boucles dans une même instruction For.

Info

Faire

Ayez garde de bien associer, dans vos boucles imbriquées, une instruction `Next` *à chaque instruction* `For`*. Chaque* `Next` *renvoie au* `For` *le plus récent. Si le* `Next` *de la boucle intérieure apparaît après le* `Next` *de la boucle extérieure, Visual Basic génère une erreur. Si vous ne spécifiez pas de variable pour chaque* `Next`*, Visual Basic se réfère automatiquement au* `For` *le plus récent. Mais, en indiquant la variable, vous circonscrirez de façon plus claire le corps de chaque boucle, et faciliterez d'autant la documentation du code.*

En résumé

Ce chapitre vous a présenté les structures de contrôle qui permettent au code de modifier lui-même le cours de son exécution en fonction des valeurs relevées et de conditions prédéfinies. Grâce aux opérateurs conditionnels et aux instructions `If`, vous pouvez maintenant analyser les données et tester les variables et contrôles, puis réagir en conséquence. Votre vocabulaire Visual Basic commence à sérieusement s'enrichir.

Outre l'instruction `If`, Visual Basic supporte l'instruction `Select Case`, qui offre une solution plus satisfaisante que l'imbrication des `If`. `Select Case` peut être utilisé sous plusieurs formats, selon la façon dont les différentes conditions doivent être testées.

Nous avons également découvert que les boucles permettent de répéter plusieurs fois certaines sections du code. La boucle `Do` se répète tant qu'une condition est remplie, ou jusqu'à ce qu'une condition soit remplie, selon le format utilisé. La boucle `For` se répète un nombre prédéfini de fois, ou jusqu'à ce qu'une condition soit remplie. A la différence de `Do`, la boucle `For` met automatiquement à jour sa variable de contrôle, en l'augmentant ou en la diminuant à chaque itération.

Le prochain chapitre vous enseigne à capter des informations à l'aide des zones d'entrée. Vous apprendrez également à donner des réponses à l'utilisateur par l'intermédiaire des boîtes de messages. Zones d'entrée et boîtes de messages constituent un moyen simple d'interagir avec l'utilisateur, sans passer par les contrôles de la feuille.

Questions-réponses

Q Pourquoi éviter l'opérateur `Not` ?

R Il vaut mieux ne pas utiliser `Not` parce que cet opérateur ne fait, dans la plupart des cas, que compliquer inutilement les choses. Les instructions positives sont toujours préférables, parce que plus faciles à comprendre.

Considérez l'expression `Not(A <= B)`. Ne serait-il pas plus simple de l'écrire `(A > B)` ? Il ne s'agit pas de proscrire tout à fait cet opérateur, utile pour interroger un booléen ; par exemple : `If Not(blnClearedScreen)`. Ce début d'instruction `If` dit ceci : "Si l'écran n'a pas encore été vidé (`blnClearedScreen = True`), alors..." En règle générale, il convient toutefois de renverser les expressions `Not` en expressions positives afin que le code soit plus clair.

Q Si les instructions `Do` et `For` donnent des résultats équivalents, quelle importance de choisir l'une ou l'autre ?

R Cela dépend entièrement de vous. Le choix ne doit d'ailleurs pas seulement s'opérer entre les boucle `For` et `Do`, mais aussi entre les divers formats de chacune. On se sert généralement des boucles `Do` pour incrémenter une valeur ou pour répéter des instructions un nombre déterminé de fois. Les boucles `Do`, quant à elles, sont plus appropriées aux boucles qui doivent se répéter jusqu'à ce qu'une condition soit remplie. Quand il s'agit d'incrémenter ou de décrémenter, `For` est plus facile à écrire et légèrement plus efficace que `Do`.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quel opérateur logique renvoie le résultat `True` si l'un ou l'autre des termes d'une expression est `True` ?
2. Quelle est la différence entre un opérateur conditionnel et un opérateur logique ?
3. Qu'est-ce qu'une boucle ?
4. Décrivez l'instruction d'affectation suivante :

```
intTotal = intTotal - 10
```

5. Combien de fois le code suivant exécute-t-il l'instruction `Beep` ?

```
intN = 0
Do While (intN > 0)
   intN = intN + 3
   Beep
Loop
```

6. Pourquoi l'instruction `Exit For` doit-il faire partie d'une instruction `If` plutôt que d'apparaître pour lui-même dans le corps de la boucle `For` ?
7. Dans une instruction `If... Else`, les deux corps peuvent s'exécuter. Vrai ou faux ?
8. Selon les valeurs initiale et finale, une boucle `For` peut ne jamais s'exécuter. Vrai ou faux ?
9. Pourquoi imbriquer des boucles `For` ?
10. Quelle est la différence entre une instruction de décision et une instruction de boucle ?

Exercices

1. Ecrivez une instruction `If` de comparaison qui vérifie l'égalité de trois nombres.
2. **Chasse au bogue :** Maurice n'arrive pas à "boucler". Qu'est-ce qui ne va pas dans son code ?

```
intN = 10
Do
  Beep
Loop Until (intN > 100)
```

3. L'horloge d'un stade de football effectue un décompte de 45 à 0 pour chaque mi-temps. Il y a deux mi-temps. Décrivez l'activité de l'horloge à l'aide d'instructions Visual Basic.
4. Visual Basic permet de combiner chaque format de `Select Case` dans une même instruction `Select Case`. On peut donc aligner un `Case` d'égalité classique, un `Case` conditionnel et un `Case` d'ordre séquentiel. Réécrivez le Listing 6.5 (où l'on calculait le paiement des heures supplémentaires à l'aide d'une instruction `If`) sous la forme `Select Case`, en utilisant au moins deux formats de `Case`.

Chapitre 7

Support avancé du clavier et de l'écran

Ce chapitre vous apprend à recevoir des informations de l'utilisateur (entrées) et à lui en envoyer (sorties). Vous savez déjà recevoir des entrées par l'intermédiaire des zones de texte, et afficher des messages dans des labels. Toutefois, ces contrôles ne se prêtent pas toujours très bien aux échanges de questions et de réponses lors de l'exécution. Les zones de texte sont parfaites pour les formulaires et les espaces réservés qui accueillent le texte tapé par l'utilisateur ; mais une interaction plus immédiate est parfois requise. Au terme de ce chapitre, vous serez en mesure d'obtenir une telle interaction en programmant le moins possible.

Voici ce que nous découvrirons aujourd'hui :

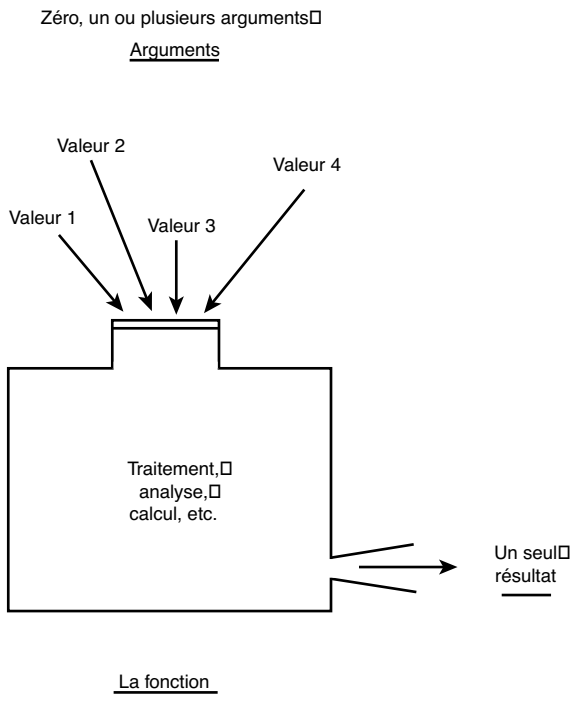
- Les fonctions internes ;
- La fonction `MsgBox()` ;
- Les paramètres de fonction optionnels ;
- La fonction `InputBox()` ;
- La gestion des événements clavier ;
- Les cases à cocher ;
- Les boutons d'option ;
- Comment combiner les boutons d'option en contrôles Frame.

Introduction aux fonctions internes

Les fonctions sont un type de procédure assez proche des sous-routines, étudiées au Chapitre 3. Elles en diffèrent toutefois en ceci que les fonctions, au terme de l'exécution de leur code, envoient une valeur vers une autre partie du programme. Ce n'est qu'au chapitre suivant que vous apprendrez à écrire des fonctions et à en analyser le fonctionnement.

Pour l'heure, il s'agit de bien comprendre les *fonctions internes*, dont nous nous servirons dans tout ce chapitre. Une fonction interne est un peu comme une boîte magique : elle reçoit les diverses valeurs que vous lui envoyez, et envoie en retour, ou renvoie, une valeur unique. La Figure 7.1 illustre le fonctionnement des fonctions.

Figure 7.1
La fonction reçoit zéro, une ou plusieurs valeurs, et renvoie une valeur unique.



Les fonctions internes, ou fonctions intrinsèques, sont des fonctions intégrées au langage Visual Basic, et qui effectuent une tâche spécifique, telle qu'un calcul ou une E/S. Avant d'utiliser une fonction interne, vous devez en connaître le nom exact et le format. Vous ne pouvez voir le code contenu dans une fonction interne car, comme son nom l'indique, elle est interne au langage, au même titre qu'un mot clé comme `For`.

E/S signifie entrée/sortie (en anglais, I/O pour input/output). Par ce terme générique, on désigne toute technique permettant de recevoir (entrée) des informations d'un périphérique (le clavier, par exemple) et d'émettre (sortie) des informations vers un périphérique (l'écran, par exemple).

Les fonctions sont extrêmement utiles ; mieux vous les comprendrez, plus vous serez à même d'en tirer parti. Le chapitre suivant traite la question plus à fond. Pour l'instant, retenez ceci :

- En général, on applique une ou plusieurs valeurs à la fonction ; il est rare qu'une fonction ne requière aucune valeur. Les valeurs appliquées à la fonction sont des *arguments*.

Un argument est une valeur appliquée à une fonction.

- Le nom de la fonction est toujours suivi de parenthèses (sauf, exception rare, pour les fonctions qui ne requièrent pas d'arguments).
- Les arguments doivent être placés entre les parenthèses de la fonction, et séparés par des virgules s'il y en a plusieurs.

Sans le savoir, vous avez déjà utilisé une fonction interne. (A partir de maintenant, nous parlerons de "fonctions" tout court.) Souvenez-vous : au Chapitre 2, vous affectiez une image au contrôle Image de la feuille. Voici la ligne de code utilisée (l'argument y a été réduit à sa plus simple expression afin de ne pas alourdir l'exemple) :

```
imgHappy.Picture = LoadPicture("\Happy.bmp")
```

Ici, le nom de la fonction est `LoadPicture()`. (Nous inclurons toujours les parenthèses dans le noms des fonctions présentées afin de bien les distinguer des noms de variables et de contrôles.) Cette fonction n'a qu'un seul argument : une chaîne.

Si la plupart en exigent au moins un, certaines fonctions se passent de tout argument. `LoadPicture()` requiert au minimum un argument de type chaîne ; tous les autres arguments sont optionnels.

Vous devez toujours respecter le type et l'ordre d'apparition des données requis pour une fonction spécifique. Par exemple, une fonction peut exiger deux arguments, un entier suivi d'une chaîne.

Qu'envoie ce code à la fonction `LoadPicture()` ? Une chaîne contenant un nom de fichier. Que renvoie `LoadPicture()` ? L'image contenue dans ce fichier. Dans le code du Chapitre 2, l'instruction suivante affectait cette image à la propriété `Picture` du contrôle Image. Sans la fonction `LoadPicture()`, l'image n'aurait jamais pu s'afficher à l'écran. Ce que le contrôle Image de la feuille attend, c'est une image affectée à la propriété `Picture` — pas un chemin d'accès (à moins que vous n'ayez spécifié, lors de la phase de conception, une image précise dont le chemin d'accès soit accessible dans la fenêtre Propriétés, et que Visual Basic affectera automatiquement au contrôle).

Lorsque vous utilisez la fonction `LoadPicture()`, beaucoup de choses se passent. D'abord, Visual Basic analyse les divers arguments appliqués et s'assure de leur conformité, en type et en nombre, avec les exigences spécifiques de la fonction. Puis Visual Basic s'assure que le chemin d'accès fourni pour l'image est valable. Enfin, si vous êtes en réseau, Visual Basic s'assure ensuite que vous avez bien accès au fichier. Et tout cela, Visual Basic le fait pour vous, sur simple exécution de la fonction `LoadPicture()` ! Voilà bien tout l'intérêt des fonctions : elles vous épargnent du boulot. Elles s'occupent des détails, et vous laissent vous concentrer sur le plus important : l'application elle-même.

Il existe des fonctions pour traiter les images, calculer des formules mathématiques simples, manipuler les fichiers, etc. Ce chapitre ne présente que les fonctions les plus simples — qui sont sans doute les plus révélatrices. Vous en apprendrez plus au prochain chapitre.

La fonction *MsgBox()*

Maintenant que vous saisissez mieux la nature des fonctions, nous pouvons examiner de près la fonction `MsgBox()`. `MsgBox()` est une fonction qui affiche une *boîte de message*. Comme le montre la Figure 7.2, une boîte de message contient une icône, un message, et au moins un bouton de commande. Ce bouton de commande laisse à l'utilisateur le temps de prendre connaissance du message ; lorsqu'il aura fini sa lecture, il cliquera sur le bouton.

Une boîte de message est une petite boîte de dialogue dont on se sert pour informer l'utilisateur à tout moment de l'exécution. La boîte de message peut être fermée, grâce au bouton de commande, ainsi que déplacée ; elle ne peut être redimensionnée.

Figure 7.2
La fonction MsgBox() affiche un message et laisse l'utilisateur indiquer qu'il a fini sa lecture.

Les arguments passés à la fonction `MsgBox()` déterminent l'icône qui sera affichée, le message, et le nombre de boutons de commande. Ainsi, le programmeur contrôle de près le message envoyé à l'utilisateur. Lorsque `MsgBox()` s'exécute, elle renvoie une valeur pour indiquer sur quel bouton l'utilisateur a cliqué. Si la boîte de message contient deux boutons, le programme interroge la valeur renvoyée par la fonction `MsgBox()`. Il peut alors déterminer, éventuellement à l'aide d'une instruction `If`, la suite des opérations, en se fondant sur la réponse de l'utilisateur.

Les versions antérieures de Visual Basic proposaient une instruction `MsgBox`. A la différence de `MsgBox()`, `MsgBox` ne pouvait pas interpréter le clic de l'utilisateur. Bien que l'instruction `MsgBox` soit considérée comme obsolète, elle est toujours supportée pour des raisons de compatibilité.

Voici le format de la fonction `MsgBox()` :

```
intResponse = MsgBox(strPrompt[, intStyle][, strTitle])
```

Dans ce format, deux arguments sont optionnels : `intStyle` et `strTitle`. Les arguments en italique sont fictifs et ne servent qu'à présenter la syntaxe ; à leur place, une fonction réelle contient des littéraux, des variables ou des arguments de contrôle. Nous avons quand même inclus les préfixes afin d'indiquer les types de données requis pour les arguments. Comme vous le voyez, une fonction `MsgBox()` exige toujours un argument chaîne, les deuxième et troisième arguments étant optionnels et dépendant du style de boîte de message souhaité.

`intResponse` indique le type de l'entier renvoyé par la fonction. Le premier argument est une chaîne (ou une chaîne, ou un contrôle qui contient une chaîne) spécifiant le message qui s'affichera dans la boîte de message. Le deuxième argument détermine le style des boutons. Le troisième spécifie le libellé qui apparaîtra dans la barre de titre.

Une boîte de message doit proposer au moins un bouton de commande. Le programme doit pouvoir déterminer si l'utilisateur a terminé sa lecture. Lorsque la boîte de message s'affiche, toute autre activité du programme est suspendue jusqu'à ce que l'utilisateur

clique sur l'un des boutons de commande. Dès que l'utilisateur clique, le code poursuit son exécution à la ligne qui suit.

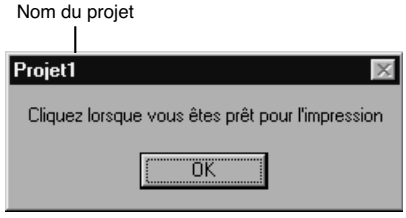
Si le message est trop long pour tenir sur une seule ligne de la boîte de message, Visual Basic procède automatiquement à la rupture de ligne (proprement et sans couper les mots).

Imaginons un programme qui attend l'ordre de l'utilisateur pour imprimer un rapport. Le message pourrait être le suivant :

```
intResponse = MsgBox("Cliquez pour lancer l'impression du rapport")
```

Ici, vous devez avoir déclaré la variable `intResponse` dans la section de déclarations de la procédure (ou bien dans la section de déclarations du module ; mais vous n'avez pas encore beaucoup d'expérience dans la déclaration des variables globales). A défaut de deuxième argument, comme c'est le cas ici, Visual Basic affiche par défaut le bouton de commande `OK` dans la boîte de message. Ainsi, puisque cette fonction `MsgBox()` n'affiche qu'un seul bouton de commande, l'affectation d'entier ne vous aide pas beaucoup. Mais la valeur renvoyée par la fonction pourra aussi bien être affectée à quelque chose.

Figure 7.3
Les fonctions MsgBox() affichent au moins un bouton de commande.



La Figure 7.3 montre également ce qui se passe si vous ne spécifiez pas tous les arguments de `MsgBox()` : Visual Basic affiche par défaut le nom du projet dans la barre de titre de la boîte de message. Il y aura toujours un nom plus approprié que celui-là ; le premier argument vous permet de le spécifier, que nous étudierons dans quelques lignes.

Pour un contrôle plus serré des boutons, vous pouvez utiliser une valeur entière (ou une variable, ou un contrôle) pour spécifier, dans le premier argument optionnel, le style des boutons. Avec un seul bouton, la valeur renvoyée ne sert pas à grand-chose, quoiqu'elle soit inévitable. Lorsque plusieurs valeurs sont en présence, la valeur renvoyée correspond au bouton sur lequel l'utilisateur a cliqué. Cette information peut être reprise dans une instruction `If` ou `Select Case`, pour que s'exécutent les sections du code chargées de la gestion de chaque bouton.

Le Tableau 7.1 présentent les valeurs utilisables comme premier argument optionnel de `MsgBox()`, pour spécifier le style du bouton.

Tableau 7.1 : Valeurs entières permettant de spécifier le style des boutons

Valeur	*Constante nommée*	*Description*
`0`	`vbOKOnly`	Bouton OK
`1`	`vbOKCancel`	Boutons OK, Annuler
`2`	`vbAbortRetryIgnore`	Boutons Abandonner, Réessayer, Ignorer
`3`	`vbYesNoCancel`	Boutons Oui, Non, Annuler
`4`	`vbYesNo`	Boutons Oui, Non
`5`	`vbRetryCancel`	Boutons Réessayer, Annuler

La Figure 7.4 montre la boîte de message générée par l'instruction suivante :

```
intResponse = MsgBox("Prêt pour l'impression ?", 1)
```

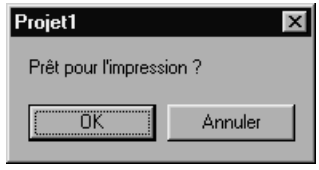


Figure 7.4
La suite du code s'exécute selon le bouton sur lequel l'utilisateur clique.

L'argument `1` spécifie que les boutons OK et Annuler doivent apparaître sur la boîte de message. Cette combinaison est utile pour les opérations que votre programme s'apprête à effectuer, par exemple imprimer un rapport, car l'utilisateur peut cliquer sur OK pour indiquer que l'imprimante est prête, ou sur Annuler pour décommander l'impression.

Le Tableau 7.2 présente les valeurs que renvoie la fonction `MsgBox()`. La boîte de message précédente pourrait être gérée par l'instruction `If` suivante (où les détails sont remplacés par des commentaires pour plus de simplicité) :

```
If (intResponse = 0) Then
   ' Ici, le code chargé de gérer
   ' le clic sur le bouton OK.
```

```
Else
    ' Ici, le code chargé de gérer
    ' le clic sur le bouton Annuler.
End If
```

Naturellement, si la boîte de message affichait d'autres boutons, l'instruction `If` *aurait à interroger des valeurs supplémentaires. Pour de multiples valeurs renvoyées, une instruction* `Select Case` *serait probablement plus appropriée.*

Tableau 7.2 : Les valeurs renvoyées indiquent quels boutons ont été cliqués

Valeur	*Constante nommée*	*Description*
1	`vbOK`	L'utilisateur a cliqué sur OK
2	`vbCancel`	L'utilisateur a cliqué sur Annuler
3	`vbAbort`	L'utilisateur a cliqué sur Abandonner
4	`vbRetry`	L'utilisateur a cliqué sur Réessayer
5	`vbIgnore`	L'utilisateur a cliqué sur Ignorer
6	`vbYes`	L'utilisateur a cliqué sur Oui
7	`vbNo`	L'utilisateur a cliqué sur Non

Si l'utilisateur appuie sur la touche Echap, `MsgBox()` *renvoie la même valeur et Visual Basic réagit de la même façon que s'il avait cliqué sur le bouton Annuler.*

Quel que soit le nombre de boutons affichés sur une boîte de message, l'utilisateur ne peut cliquer que sur un seul. Dès qu'il clique sur l'un des boutons, la boîte de message se referme, et `MsgBox()` renvoie la valeur correspondante.

Les constantes nommées

Remarquez, aux Tableaux 7.1 et 7.2, la colonne "Constantes nommées". Visual Basic supporte des centaines de *constantes nommées*, utilisées dans les procédures à la place des littéraux.

Les constantes nommées sont des noms internes à Visual Basic, qui correspondent à des valeurs prédéfinies. Les constantes nommées commencent généralement par le préfixe `vb`*. A la différence des variables déclarées, les constantes nommées ne peuvent changer de valeur (d'où le nom de constantes). Mais vous pouvez recourir aux constantes nommées comme arguments d'une fonction, à la place des littéraux.*

Les constantes nommées rendent les programmes plus lisibles et plus compréhensibles. Par exemple, les instructions suivantes sont équivalentes, mais, dans la seconde, la mention du bouton de commande est explicite :

```
intResponse = MsgBox("Prêt pour l'impression ?", 1)
intResponse = MsgBox("Prêt pour l'impression ?", vbOK)
```

Vous pouvez, dans le cours de l'écriture, recourir aux constantes nommées sans avoir à consulter un manuel de référence ni l'aide en ligne, et sans avoir à mémoriser des noms abracadabrants. Pour chaque fonction que vous utilisez, Visual Basic propose automatiquement une liste des constantes nommées possibles pour cette fonction. Lorsque, par la suite, vous maintiendrez ou modifierez votre programme, vous devinerez sans peine le style de la boîte de message. Si, au lieu de constantes nommées, votre code présente des valeurs littérales, il faudra vous remémorer le sens de ces valeurs avant de modifier la boîte de message.

Utilisez les constantes nommées partout où c'est possible. Elles ne réclament pas plus de saisie, puisque Visual Basic vous propose une liste contextuelle à mesure que vous entrez les arguments.

Les boutons par défaut

Le premier bouton d'une boîte de message est toujours le bouton par défaut. Visual Basic met automatiquement le focus sur le premier bouton (le plus à gauche), et le déclenche dès que l'utilisateur appuie sur Entrée.

Vous pouvez intervenir sur l'ordre d'apparition des boutons sur la boîte de message. Il suffit, pour cela, d'ajouter au bouton de commande les arguments adéquats (voir Tableau 7.3).

En dépit de leur longueur, les constantes nommées sont plus faciles à maintenir que les littéraux. Considérez l'instruction suivante :

```
intResponse = MsgBox("L'imprimante est-elle allumée ?",
➥vbYesNoCancel + vbDefaultButton1)
```

Tableau 7.3 : Ces arguments permettent de spécifier le bouton par défaut

Valeur	*Constante nommée*	*Description*
`0`	`vbDefaultButton1`	Premier bouton par défaut
`256`	`vbDefaultButton2`	Deuxième bouton par défaut
`512`	`vbDefaultButton3`	Troisième bouton par défaut

Quand la boîte de message autorise une opération risquée, comme la suppression d'un fichier, il est recommandé de spécifier Annuler comme bouton par défaut. Ainsi, si l'utilisateur appuie sur Entrée par erreur, le bouton Annuler est déclenché, et l'opération n'est pas effectuée ; ce qui aurait été le cas si OK était resté le bouton par défaut.

Les icônes

En adjoignant une valeur supplémentaire au deuxième argument, vous spécifiez l'icône qui apparaîtra à gauche du message. Comme nous n'avons pas encore touché à cette partie de l'argument, nos boîtes de message n'affichaient pas d'icône.

En vérité, la fonction `MsgBox()` supporte quelques autres arguments optionnels. Mais, rarement utilisés dans les programmes simples, ils ne seront pas présentés ici.

Le Tableau 7.4 présente les constantes nommées utilisables et les icônes correspondantes.

Tableau 7.4 : Valeurs spécifiant l'icône de la boîte de message

Valeur	*Constante nommée*	*Description*	*Icône*
`16`	`vbCritical`	Erreur critique	
`32`	`vbQuestion`	Question	

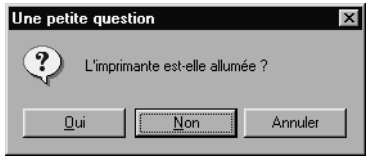
Tableau 7.4 : Valeurs spécifiant l'icône de la boîte de message *(suite)*

Valeur	*Constante nommée*	*Description*	*Icône*
48	vbExclamation	Avertissement	
64	vbInformation	Information	

Les instructions suivantes génèrent une boîte de message complète. Tous les arguments sont spécifiés, donc tous les éléments apparaissent. Le résultat est reproduit en Figure 7.5.

```
intResponse = MsgBox("L'imprimante est-elle allumée ?", vbYesNoCancel
➥+ vbQuestion + vbDefaultButton2, "Une petite question")
```

Figure 7.5
Une boîte de message complète.



La fonction *InputBox()*

La fonction `MsgBox()` permet d'envoyer des messages à l'utilisateur, et donne la possibilité à celui-ci de répondre en cliquant sur des boutons de commande. La suite du code s'exécute en fonction du bouton sélectionné. Naturellement, s'il ne s'agit que d'un message d'information, un seul bouton de commande est requis, pour que l'utilisateur puisse refermer la boîte après consultation.

Il arrive que la question à poser soit si simple, et exige une réponde si rapide, qu'un contrôle zone de texte serait absolument hors de propos. On peut alors faire appel à la cousine de `MsgBox()` : `InputBox()`. La fonction `InputBox()` affiche une boîte de message qui permet à l'utilisateur de répondre "dans le texte". Cette combinaison d'une boîte de message et d'une sorte de zone de texte est appelée *boîte d'entrée*. La Figure 7.6 en montre un exemple.

Une boîte d'entrée est une boîte de message qui inclut un champ. Dans ce champ, l'utilisateur peut saisir la réponse à une question qui lui est posée. Tout comme les boîtes de message, les boîtes d'entrée sont déplacées et fermées, mais pas redimensionnées. Contrairement aux boîtes de message, en revanche, les boîtes d'entrée ne vous laissent pas le choix des boutons de commande. Seuls les boutons OK et Annuler apparaissent dans une boîte d'entrée.

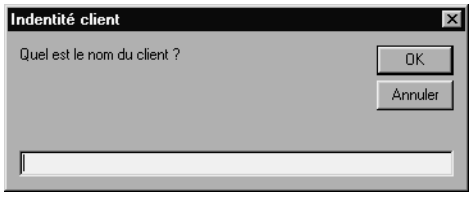


Figure 7.6
La boîte d'entrée affiche un titre, un message et un champ de saisie.

Les boîte d'entrée n'affichent pas non plus d'icône, contrairement aux boîtes de message. Voici le format de la fonction `InputBox()` :

```
strAnswer = InputBox(strPrompt[, strTitle][, strDefault][, intXpos]
➥[, intYpos])
```

`InputBox()` renvoie une valeur de type `Variant`, qui peut toujours être traitée comme une chaîne. La fonction `InputBox()` peut donc être affectée à une chaîne, et utilisée comme une valeur de chaîne. (Le type `Variant` autorise également l'affectation de la valeur renvoyée à une propriété de contrôle.) La chaîne renvoyée est la réponse saisie par l'utilisateur dans le champ. Seul le premier argument est requis. Voici une description de tous les arguments :

- *`strPrompt`*. Le message ou la question (*prompt*, en anglais) qui s'affiche dans la boîte d'entrée. La longueur maximale de *`strPrompt`* est de 1 024 caractères. Tourner toujours le message ou la question de sorte que l'utilisateur sache quoi répondre.
- *`strTitle`*. Le texte qui apparaît dans la barre de titre. A défaut de titre, Visual Basic affiche le nom du projet.
- *`strDefault`*. Contenu par défaut du champ de saisie. L'utilisateur peut accepter cette réponse par défaut, qui sera alors la valeur renvoyée ; il peut la modifier, ou saisir une réponse entièrement nouvelle. On se sert d'une valeur par défaut lorsque la réponse est prévisible et n'attend que d'être validée par le bouton OK.
- *`intXpos`*, *`intYpos`*. Coordonnées en twips de la boîte d'entrée. Il est préférable, quand la question posée renvoie à d'autres feuilles ou boîtes de dialogue déjà à

l'écran, de ne pas afficher la boîte d'entrée par-dessus. Si vous ne spécifiez pas de coordonnées, Visual Basic affiche par défaut la boîte d'entrée au centre de l'écran.

Un twip vaut 1/567 de centimètre et 1/1440 de pouce.

Voici l'instruction qui a généré la boîte d'entrée de la Figure 7.6 :

```
strAnswer = InputBox("Quel est le nom du client ?",
➥"Indentité client")
```

Pour afficher une valeur par défaut et positionner la boîte d'entrée à un point précis de l'écran, l'instruction serait :

```
strAnswer = InputBox("Quel est le nom du client ?",
➥"Indentité client", "Jean Bon", 500, 750)
```

Le programme doit pouvoir déterminer si l'utilisateur a cliqué sur OK (ou appuyé sur Entrée, OK étant le bouton par défaut) ou s'il a cliqué sur Annuler. Si l'utilisateur clique sur Annuler au lieu d'entrer une nouvelle valeur ou de valider par OK la valeur par défaut, la fonction `InputBox()` renvoie une chaîne nulle, c'est-à-dire `""`.

L'interrogation du résultat pourrait alors prendre la forme suivante :

```
If (strAnswer <> "") Then
    ' Code chargé de gérer l'entrée utilisateur
Else
    ' Code chargé de gérer le clic sur Annuler
End If
```

Rappelez-vous que Visual Basic supporte la valeur spéciale `Empty`, qui peut prendre la place de `""` dans l'instruction. Avec le mot clé `Empty`, le code est plus clair. Notre instruction `If` ressemblerait alors à ceci :

```
If (strAnswer <> Empty) Then
```

Imaginons que l'utilisateur cherche à calculer le chiffre de vente total d'un agent commercial particulier. Votre programme affiche une boîte d'entrée pour demander le nom de l'employé. Si l'utilisateur saisit le nom et valide sa réponse par OK, le code correspondant s'exécute et calcule le chiffre de vente demandé. Si, en revanche, l'utilisateur choisit Annuler, le programme ignore le code chargé du calcul.

Gestion du clavier

Les contrôles et boîtes d'entrée ne peuvent suffire à traiter toutes les entrées clavier. Le programme doit être en mesure de répondre à des touches spécifiques au moment où l'utilisateur les frappe. Comme nous l'avons vu, Windows passe à votre programme les éléments qui sont de son ressort afin qu'il puisse les traiter. Il s'agit des événements `KeyPress`, `KeyDown` et `KeyUp`. Ils répondent à des combinaisons de touches du genre Alt-G ou Maj-P, ainsi qu'aux touches individuelles. Lorsqu'un événement clavier se produit, ces combinaisons sont testées.

Une fois que l'application reçoit une entrée clavier, elle modifie cette entrée, ou bien l'ignore s'il ne s'agit pas de la frappe attendue. Le traitement des événements clavier permettent de déclencher la fermeture d'un écran de démarrage, de valider une entrée, de jouer à des jeux, etc.

Les événements clavier

L'événement `KeyPress` a lieu lorsque l'utilisateur appuie sur une quelconque touche parmi les suivantes :

- Lettres capitales et minuscules ;
- Chiffres ;
- Signes de ponctuation ;
- Entrée, Tab et Retour arrière.

L'événement `KeyPress` reconnaît la plupart des caractères ASCII. Font toutefois exception la touche de tabulation, les touches flèches et autres caractères spéciaux dont le code ASCII est compris entre 0 et 31. `KeyPress` permet de déterminer avec exactitude la touche que l'utilisateur a frappée. Si l'utilisateur appuie sur la touche "A", `KeyPress` renvoie "A", etc.

L'événement `KeyPress` a lieu lorsque la touche est enfoncée. Si l'utilisateur laisse son doigt dessus, l'événement ne se répète que si le clavier est paramétré en mode "refrappe".

Un événement, nous l'avons vu, est toujours associé à un objet tel qu'un bouton de commande ou une feuille. L'événement `KeyPress` est associé à l'objet qui, au moment de la frappe, a le focus. Si aucun objet n'a le focus, `KeyPress` s'applique à la feuille. (La propriété `KeyPreview` induit une exception à cela, que nous expliquerons dans la section sur la priorité des réponses, plus loin dans ce chapitre.)

Les événements clavier ne doivent pas être utilisés comme des raccourcis clavier pour les menus. Le Créateur de menus s'occupe d'attribuer les raccourcis clavier selon vos spécifications, et gère automatiquement la réponse en déclenchant la procédure événementielle `Click`*. Dans de telles conditions, l'interrogation des événements clavier empêcherait le programme de répondre aux sélections de menu.*

Les procédures événementielles `KeyPress` incluent toujours un argument de type `Integer`. Si vous deviez écrire une procédure événementielle `KeyPress` pour un contrôle zone de texte, voici ce que cela donnerait :

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
  '
  ' Ici, le code chargé d'interroger
  ' et de traiter les événements clavier.
  '
End Sub
```

L'argument `KeyAscii` est un entier correspondant au code ASCII de la touche frappée. Vous pouvez, à l'aide d'instructions `If` ou `Select Case`, vérifier qu'il s'agit bien de la touche attendue.

`KeyPress` effectue une autre tâche très utile : changer l'entrée utilisateur. A proprement parler, l'événement `KeyPress` se produit *entre* le moment où l'utilisateur frappe la touche et le moment où le contrôle destinataire reçoit la valeur. Bien entendu, cet "intermède" demeure imperceptible à l'exécution. Ainsi, un contrôle zone de texte qui a le focus affiche immédiatement le caractère correspondant à la touche frappée. Mais la procédure événementielle `KeyPress` de ce contrôle peut aussi bien changer le caractère en cours de route, comme le montre le code suivant :

```
Private Sub txtTryIt_KeyPress(KeyAscii As Integer)
  ' Changer le caractère "A" en caractère "B".
  If KeyAscii = 65 Then   ' 65 est le code ASCII pour "A".
    KeyAscii = 66         ' 66 est le code ASCII pour "B".
  End If
End Sub
```

Si le contrôle `txtTryIt` a le focus, la zone de texte accepte et affiche les caractères que l'utilisateur entre au clavier — à une exception près. Car l'instruction `If` change la valeur `KeyAscii` de la lettre capitale "A" (ASCII 65) en la valeur de la lettre capitale "B" (ASCII 66). Lorsque l'utilisateur tape "A", la zone de texte affiche "B". L'événement `KeyPress` intercepte la frappe avant que le contrôle zone de texte ne reçoive la valeur `KeyAscii`.

Recherchez "Key Code Constants" dans l'aide en ligne de Visual Basic. Vous y trouverez la liste des constantes nommées permettant d'interroger l'activité du clavier. On peut ainsi répondre à la touche Retour arrière en vérifiant que `KeyAscii` *vaut* `vbKeyBack`*, à la touche Entrée avec* `vbKeyReturn`*, ou à la touche Tab avec* `vbKeyTab`*. Rappelez-vous que* `KeyPress` *ne peut répondre qu'à ces trois touches, ainsi qu'aux lettres, nombres et signes de ponctuation. Si les zones de texte reconnaissent d'autres touches (telles que Origine et Fin),* `KeyPress` *n'est fiable que pour Entrée, Tab et Retour arrière.*

`KeyPress` permet d'interroger une vaste gamme de frappes. L'événement `KeyDown` est plus spécifique. Tout comme `KeyPress`, il a lieu lorsque l'utilisateur appuie sur une touche. Mais `KeyDown` offre un relevé plus détaillé — donc un peu plus complexe — de l'activité du clavier. Par exemple, `KeyPress` renvoie une valeur ASCII différente, selon que l'utilisateur frappe la capitale "T" ou la minuscule "t". `KeyDown` renvoie la *même* valeur pour les deux, mais il lui associe une autre valeur : c'est l'argument d'*état*, qui indique l'état de la touche Maj.

L'événement `KeyDown` *a lieu lorsque l'utilisateur appuie sur une touche. Les deux événements* `KeyDown` *et* `KeyPress` *peuvent donc se produire en même temps (pour une touche ASCII).*

Pour interroger une frappe ASCII, `KeyPress` *est préférable à* `KeyDown`*, parce que plus simple à programmer.*

Voici les lignes d'encadrement d'une procédure événementielle `KeyDown` :

```
Private Sub txtTryIt_KeyDown(KeyCode As Integer, Shift As Integer)
    '
    '  Ici, le code de gestion du clavier.
    '
End Sub
```

`KeyCode` représente la touche frappée, tandis que l'argument `Shift` détermine l'état de la touche Maj (qui peut aussi bien être Ctrl ou Alt). `KeyCode` renvoie toujours l'équivalent en lettre *capitale* de la touche frappée. Ainsi, même si l'utilisateur entre un "t" minuscule, l'argument `KeyCode` vaudra 84 (code ASCII de la capitale "T").

Il faut se montrer très vigilant avec `KeyDown`*, car le fait que le style soit ignoré peut entraîner des confusions. Pour la frappe d'une touche numérique, par exemple, l'argument* `Shift` *doit impérativement être interrogé. Selon que l'utilisateur ait ou non enfoncé la touche Maj (ce qu'indique l'argument*

`Shift`), le caractère à afficher sera le chiffre de la touche, ou le caractère secondaire inscrit en dessous du chiffre (par exemple, "9" avec Maj, "ç" sans Maj).

L'avantage principal de `KeyDown` sur `KeyPress` est que, tout `Shift` mis à part, `KeyDown` peut interroger *n'importe quelle* frappe incluant les touches flèches, Origine, Fin, etc. Nous vous invitons de nouveau à consulter l'aide en ligne de Visual Basic au sujet des constantes nommées relatives au clavier.

L'*état de shift* indique si une touche de contrôle (Maj, Ctrl, Alt, ou aucune des trois) a été frappée en même temps qu'une autre. Le modèle binaire interne de l'argument `Shift` détermine l'état de shift. Pour interroger l'état de shift, vous devez utiliser l'opérateur `And` avec la valeur 7. (Ce type particulier de `And` est dit *binaire — bitwise*, en anglais —, par opposition à l'opérateur logique `And` classique, qui procède à des comparaison.) Le Listing 7.1 donne un exemple d'interrogation de l'état de shift.

Listing 7.1 : Code pour tester l'état de shift

```
1:  Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)
2:    Dim intShiftState As Integer
3:    intShiftState = Shift And 7  ' "And" binaire
4:    Select Case intShiftState
5:      Case 1
6:        ' Code pour les combinaisons Maj
7:      Case 2
8:        ' Code pour les combinaisons Ctrl
9:      Case 3
10:       ' Code pour les combinaisons Alt
11:     Case 4
12:       ' Code pour les combinaisons Maj-Ctrl
13:     Case 5
14:       ' Code pour les combinaisons Maj-Alt
15:     Case 6
16:       ' Code pour les combinaisons Ctrl-Alt
17:     Case 7
18:       ' Code pour les combinaisons Maj-Ctrl-Alt
19:     End Select
20: End Sub
```

L'événement `KeyUp` a lieu lorsque l'utilisateur relâche la touche frappée. Pour vérifier la relâche d'une touche spécifique (comme la touche A, si l'utilisateur relâche la "moitié" de la combinaison Maj-A), il faut interroger les arguments passés à `KeyUp()`. `KeyUp` a donc lieu après les événements `KeyDown` et `KeyPress`.

Le code suivant montre la procédure événementielle d'une zone de texte. Il s'agit de convertir les lettres minuscules entrées par l'utilisateur en lettres capitales :

```
1:  Private Sub txtTry_KeyPress(KeyAscii As Integer)
2:    ' Convertir les minuscules en capitales
3:    If (KeyAscii >= 97) And (KeyAscii <= 122) Then
4:      KeyAscii = KeyAscii - 32   ' Passage en capitales
5:    End If
6:  End Sub
```

Comme vous pourrez le vérifier à l'Annexe C, la plage de valeurs ASCII des lettres minuscules va de 97 ("a") à 122 ("z"). La différence de valeur ASCII entre une lettre capitale donnée et son équivalent minuscule est de 32. Ainsi, dans notre code, la procédure événementielle `KeyPress` reçoit la valeur ASCII de chaque lettre minuscule tapée, et lui soustrait 32 pour obtenir la capitale correspondante.

N'utilisez pas les événements clavier pour écrire vos propres routines MaskedEdit. Contentez-vous d'appuyer sur Ctrl-T pour ajouter à la Boîte à outils le contrôle Microsoft Masked Edit Control 6.0. (Le Chapitre 9 explique plus en détail comment l'on ajoute des outils à la Boîte à outils.) Le contrôle MaskEdit permet de créer des masques de saisie, tels que des champs de numéros de téléphone dans lesquels les tirets s'affichent automatiquement. Essayez d'écrire vos propres routines à cet effet, ce serait réinventer la roue, et gaspiller un temps précieux.

L'instruction *SendKeys*

L'instruction `SendKeys` permet d'envoyer des frappes clavier *depuis le code*, exactement comme si l'utilisateur appuyait sur les touches. Par exemple, `SendKeys` est utile pour contrôler le placement du curseur texte dans une zone de texte : il suffit de "simuler" la frappe des touches Origine ou Fin. Voici la syntaxe de `SendKeys` :

```
SendKeys strKeystrokes[, blnWait]
```

`strKeystrokes` sera un littéral chaîne, tel que "Gibolin SARL", dans les cas où vous entrez les valeurs à la place de l'utilisateur. L'option *`blnWait`* est généralement omise. Il s'agit d'un booléen qui, s'il renvoie `False` (valeur par défaut si *`blnWait`* est omis), s'assure que le contrôle revient à la procédure en exécution dès que les frappes sont reçues. Si *`blnWait`* est `True`, le système traite les frappes avant de poursuivre l'exécution du code, de sorte que les événements clavier sont actifs pendant la frappe.

Dans une instruction `SendKeys`, il faut entourer d'accolades (`{` et `}`) les caractères suivants : `^`, `+`, `%`, `~` (tilde) et les parenthèses. Ainsi, pour envoyer *via* `SendKeys` la chaîne `7 + 6`, il faudrait écrire :

```
SendKeys "7 {+} 6"
```

Les touches de fonction et les touches spéciales, comme Origine, doivent être envoyées par `SendKeys` et entre accolades. Par exemple, pour envoyer au programme l'équivalent d'une frappe sur la touche Origine, vous devrez utiliser le littéral `{Home}`, comme suit :

```
SendKeys "{Home}"
```

Recherchez `SendKeys` dans l'aide en ligne de Visual Basic pour connaître les constantes nommées affectées à chaque touche spéciale.

`SendKeys` ne permet pas d'envoyer la touche Impr écran.

Priorité des réponses

Lorsque l'utilisateur appuie sur une touche, la feuille ou le contrôle qui a le focus reçoit la frappe. Si aucun contrôle n'a le focus, c'est la feuille qui capte l'événement clavier. Si, en revanche, un contrôle a le focus, l'événement peut lui être envoyé comme à la feuille, selon la spécification de la propriété `KeyPreview` de la feuille.

Si sa propriété `KeyPreview` est `True`, la feuille reçoit l'événement clavier. Si donc vous avez écrit deux procédures événementielles `frmAcct_KeyDown()` et `txtEntry _KeyDown()`, et que la propriété `KeyPreview` de la feuille soit `True`, `frmAcct_KeyDown()` s'exécute lorsque l'utilisateur appuie sur une touche. Si la propriété `KeyPreview` de la feuille est `False`, c'est `txtEntry_KeyDown()` qui s'exécute (si tant est que la zone de texte `txtEntry` ait le focus à ce moment-là).

Contrôles supplémentaires

Boîtes de message et boîtes d'entrée offrent un moyen commode d'envoyer et de recevoir des informations, par l'intermédiaire de fenêtres qui s'ouvrent et se ferment à volonté. Les boîtes de message et d'entrée forment un complément intéressant aux labels et zones de texte, parce qu'elles affichent et recueillent les entrées d'une manière différente.

Il vous reste à découvrir d'autres contrôles, qui permettent de recevoir des entrées et laissent à l'utilisateur plusieurs choix. Ce sera l'objet de la suite du chapitre. Au terme de votre lecture, vous serez en mesure d'intégrer à vos applications plusieurs nouveaux contrôles.

Les cases à cocher

La *case à cocher* propose une option à l'utilisateur. Une case à cocher peut apparaître seule ou au milieu d'autres cases à cocher. Lorsque l'utilisateur clique dessus, la case se coche pour indiquer que l'option est sélectionnée. Si l'utilisateur clique de nouveau sur la case, l'option est désélectionnée et la coche disparaît.

Une case à cocher (check box, en anglais) est une option proposée sur la feuille. La case est cochée lorsque l'option est sélectionnée, décochée lorsque l'option est désélectionnée. Les cases à cocher offrent à l'utilisateur le choix entre deux valeurs possibles : `True`/`False`, `Enabled`/`Disabled`, *etc.*

Une case à cocher est soit cochée, soit décochée. `Value` est la propriété du contrôle `CheckBox` qui détermine l'état de la case. Si la propriété `Value` vaut `1`, l'option est sélectionnée et la case est cochée. Si `Value` vaut `0`, l'option est désélectionnée et la case décochée.

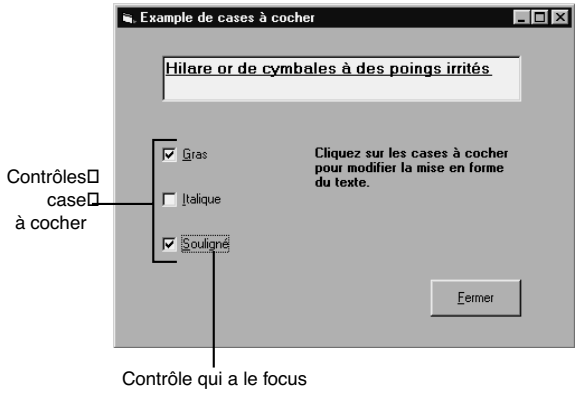
Les valeurs `1` *ou* `0` *de la propriété* `Value` *correspondent respectivement à des valeurs* `True` *ou* `False` *pour un même contrôle case à cocher. Si vous attendez de l'utilisateur qu'il réponde par oui ou non, inutile de lui fournir une case à cocher pour chaque réponse. D'ailleurs, les cases à cocher servent plus à indiquer qu'une option est sélectionnée qu'à donner une réponse oui ou non. La Figure 7.7 représente une feuille contenant trois cases à cocher. Chaque contrôle peut être coché ou décoché, selon les sélections de l'utilisateur et selon les valeurs par défaut de la propriété* `Value`*, spécifiées par le programmeur lors de la conception.*

En affectant un raccourci clavier (tel que Alt-G) à la propriété `Caption` *de la case à cocher, vous permettez à l'utilisateur de sélectionner ou de désélectionner l'option à partir du clavier.*

L'état d'une case à cocher peut être interrogé par une instruction `If` du format suivant :

```
If (chkUnder.Value = 1) Then
    ' Code chargé de gérer l'état "coché"
Else
    ' Code chargé de gérer l'état "décoché"
End If
```

Figure 7.7
Les cases à cocher offrent à l'utilisateur le choix entre diverses options.

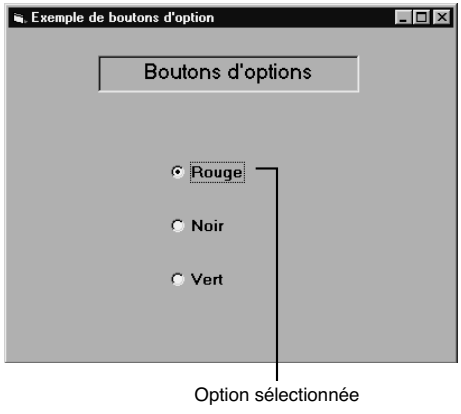


Rappelez-vous que votre feuille peut contenir plusieurs cases à cocher. Il est possible, par une programmation fastidieuse, de faire en sorte qu'une seule case soit cochée à la fois. Mais Visual Basic dispose d'un moyen plus simple et plus efficace pour offrir des options mutuellement exclusives. C'est ce que nous allons maintenant découvrir.

Les boutons d'option

Les *boutons d'option* donnent à l'utilisateur la possibilité de choisir parmi plusieurs options. A l'inverse des cases à cocher, en revanche, un seul bouton d'option peut être sélectionné à la fois. La Figure 7.8 montre une feuille contenant trois boutons d'option, dont un seul est sélectionné. Si l'utilisateur clique sur un bouton d'option, Visual Basic désélectionne automatiquement tous les autres.

Figure 7.8
Les boutons d'option offrent un choix unique parmi plusieurs options.



Un bouton d'option ne permet à l'utilisateur de choisir qu'une seule option à la fois. Les boutons d'option sont parfois nommés "boutons radio", par analogie avec le système de boutons des anciens autoradios.

La procédure événementielle `Form_Load()` vous permet de définir, à l'exécution, la propriété `Value` de tous les boutons comme `False`. Lorsque la feuille s'affiche, aucun des boutons d'option ne sera sélectionné. Mais, dès que l'utilisateur clique sur un bouton d'option, ce bouton reste sélectionné jusqu'à ce que l'utilisateur en sélectionne un autre. Si vous affectez un raccourci clavier à la propriété `Caption` des boutons d'option, l'utilisateur pourra sélectionner l'option correspondante sans se servir de la souris.

Ne placez jamais un bouton d'option tout seul sur une feuille, car ce bouton, une fois sélectionné, ne pourrait plus être désélectionné.

Le contrôle Frame et les groupes d'options

L'utilisateur peut sélectionner plusieurs boutons d'option à la fois sur une même feuille, à condition que ces boutons résident dans des zones séparées, appelées *frames*. Les frames regroupent les boutons d'option.

Le frame, parfois dit contrôle conteneur, regroupe des contrôles dans une zone distincte de la feuille elle-même. Vous pouvez ainsi proposer dans une même fenêtre plusieurs jeux de boutons d'option, chaque jeu étant bien circonscrit dans son frame. Les frames ne s'appliquent pas, d'ailleurs, qu'aux boutons d'option ; ils peuvent recevoir tous les types de contrôles que vous voudrez regrouper.

La Figure 7.9 montre une feuille dont deux boutons d'option sont sélectionnés. Cela n'est possible que parce que la feuille contient d'un côté trois boutons, de l'autre un frame avec deux autres boutons. Sans le frame, un seul des cinq boutons d'option pourrait être sélectionné.

Faire
N'hésitez pas à placer sur la feuille autant de frames que nécessaire afin d'offrir différentes catégories d'options dans une même fenêtre.

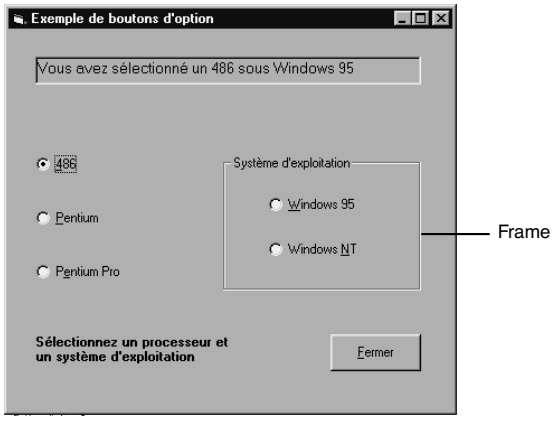


Figure 7.9
En plaçant un frame sur la feuille, vous permettez à l'utilisateur de sélectionner un bouton dans chaque groupe.

Les frames sont très faciles à créer. Voici les propriétés utiles à cet effet :

- `BorderStyle`. Les deux valeurs possibles sont `0-None` et `1-Fixed single`. Comme pour la plupart des propriétés de ce type, vous pouvez affecter les valeurs `0` ou `1` au moyen d'instructions d'affectation, à l'exécution, ou bien spécifier une valeur par défaut dans la fenêtre Propriétés, lors de la conception. Avec la valeur `0`, le frame ne présentera pas de bordure ni de libellé, et la séparation d'avec le reste de la feuille sera donc invisible. Un frame sans bordure reste un frame, et les boutons d'option qu'il contient restent indépendants des autres boutons d'option de la feuille ; mais l'utilisateur éprouvera quelque difficulté à discerner les différents groupes.
- `Caption`. Définit le libellé qui apparaîtra en haut du frame.
- `Font`. Définit les attributs de police du contenu de `Caption`.

Avant d'attaquer le chapitre suivant, nous vous proposons un troisième Projet bonus, "Entrées utilisateur et logique conditionnelle". Ce projet vous invite à créer une application complète, incluant notamment des boutons d'option regroupés en frame. Vous apprendrez ainsi à disposer proprement les frames sur la feuille, et à y intégrer les divers boutons d'option. En effet, pour que Visual Basic reconnaisse que les boutons d'option font partie du frame et non de la feuille elle-même, vous devez les placer par-dessus le frame.

En résumé

Ce chapitre vous a présenté les fonctions internes. Vous en découvrirez de nouvelles en progressant dans cet ouvrage. Il suffit d'appeler les fonctions, de leur appliquer des arguments et de traiter les valeurs renvoyées pour effectuer des opérations complexes sans avoir à écrire de codes fastidieux.

La fonction `MsgBox()` permet d'adresser des messages à l'utilisateur par le biais de petites fenêtres. En guise de réponse, l'utilisateur ne peut que cliquer sur un bouton de commande. Les boîtes de message permettent notamment de demander l'avis de l'utilisateur avant d'effectuer une opération quelconque ; pour décommander l'opération, il lui suffit de cliquer sur Annuler. Là où la fonction `MsgBox()` se contente d'afficher des messages, la fonction `InputBox()` permet de poser des questions à l'utilisateur et de recevoir des réponses dans une même fenêtre.

Le programme peut également obtenir des informations de l'utilisateur par le biais d'autres contrôles, tels que les cases à cocher ou les boutons d'option (lesquels peuvent être regroupés en frames). Le code s'exécute en fonction des options sélectionnées par l'utilisateur.

Le prochain chapitre pousse un peu plus loin l'étude de la structure des programmes Visual Basic. Vous approfondirez la notion de variables locales et globales, et découvrirez à peu près toutes les fonctions internes que vous pouvez utiliser.

Questions-réponses

Q Pourquoi une fonction interne ne peut-elle renvoyer plus d'une valeur ?

R On pourrait dire que la fonction interne *devient* sa valeur renvoyée. Une fonction interne marche comme une expression : elle produit une valeur, et une seule. On passe à la fonction interne une valeur ou un ensemble de valeurs qu'elle doit traiter ou combiner d'une façon ou d'une autre. La valeur renvoyée est le résultat de ce traitement ou de cette combinaison. Par exemple, la fonction reçoit un argument chaîne qui spécifie le chemin d'accès d'une image ; la valeur renvoyée n'est autre que cette image elle-même, qui pourra être affectée à un contrôle ou à une propriété graphique.

De par leur nature, les fonctions internes peuvent être utilisées partout où peuvent être utilisées les valeurs qu'elles renvoient. Ainsi, plutôt que d'afficher un littéral chaîne ou une variable dans une boîte de message, vous pouvez appeler une fonction `InputBox()` en lieu et place de la chaîne à afficher. En procédant ainsi, vous imbriquez une fonction dans une autre. La fonction intérieure, `InputBox()`, s'exécute en

premier et obtient une chaîne de l'utilisateur ; puis `MsgBox()` s'exécute à son tour et affiche la chaîne reçue dans une boîte de message. Voici un exemple :

```
intResp = MsgBox(InputBox("Quel est votre nom ?"))
```

Quand aurez-vous à imbriquer une fonction `InputBox()` dans une fonction `MsgBox()` ? Peut-être jamais. Mais cette instruction d'affectation montre bien que la fonction interne, ici `InputBox()`, "devient" en quelque sorte sa valeur renvoyée, et peut être immédiatement utilisée comme telle dans le code.

Q Quelles sont les autres fonctions internes disponibles ?

R Des fonctions internes sont disponibles pour traiter des nombres, des chaînes, et autres types de données. Vous les découvrirez dans le chapitre suivant, "Sous-routines et fonctions".

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront la mise en pratique de ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quelle est la différence entre un *argument* et une *fonction interne* ?
2. On peut spécifier le bouton par défaut d'une boîte de message. Vrai ou faux ?
3. Quel mot clé est équivalent à la chaîne nulle `""` ?
4. Les Tableau 7.1, 7.2 et 7.3 décrivent trois arguments différents qui peuvent être utilisés avec la fonction `MsgBox()`. Vrai ou faux ?
5. Qu'apparaît-il par défaut dans la barre de titre des boîtes de message et d'entrée si vous ne spécifiez pas l'argument correspondant ?
6. Quelle est la différence essentielle entre une case à cocher et un bouton d'option ?
7. On peut afficher des boutons d'option sur une feuille sans qu'un seul ne soit sélectionné. Vrai ou faux ?

8. Quelle valeur de propriété détermine si une case à cocher est sélectionnée ou non ?
9. Quelle valeur de propriété détermine si un bouton d'option est sélectionné ou non ?
10. Pourquoi est-il parfois nécessaire d'inclure des boutons d'option dans un frame ?

Exercices

1. Décrivez de quelle façon le code parvient à déterminer si l'utilisateur a saisi une valeur dans une boîte d'entrée (ou accepté la valeur par défaut), ou s'il a cliqué sur le bouton Annuler.
2. Ecrivez la fonction `MsgBox()` nécessaire pour générer la boîte de message de la Figure 7.10.

Figure 7.10
Comment générer cette boîte de message ?

3. Ecrivez une procédure événementielle de bouton de commande qui demande à l'utilisateur sa ville puis son département, cela dans deux boîtes d'entrée séparées. Ensuite, concaténez les deux chaînes renvoyées en intercalant une virgule et une espace entre les deux. Enfin, affichez la chaîne résultante dans une boîte de message.
4. Créez une feuille contenant cinq boutons d'option, destinés à simuler de véritables "boutons radio". A chaque bouton, attribuez un libellé figurant le nom d'une de vos cinq stations de radio préférées. Ecrivez une procédure événementielle pour chaque contrôle, de sorte que la sélection d'un bouton d'option provoque l'affichage d'une boîte de message qui décrive (style de musique, etc.) la radio en question.

PB3

Entrées utilisateur et logique conditionnelle

Ce Projet bonus met en pratique les différents contrôles que nous avons étudiés, ainsi que la gestion des réponses utilisateur. L'application que vous allez créer inclura des cases à cocher, des boutons d'option et des frames.

Tout cela va naturellement demander un peu de programmation. En fait, ce Projet bonus contient plus de code que vous n'en avez encore vu. Voici les objectifs de l'application :

- Proposer une série de cases à cocher qui permettent à l'utilisateur de choisir un ou plusieurs pays, et d'afficher les drapeaux correspondants.
- Proposer les pays, mais avec, cette fois, des boutons d'option, de sorte que l'utilisateur ne puisse sélectionner qu'un seul pays à la fois.
- Générer un avertissement, sous forme de boîte de message, quand l'utilisateur n'entre pas l'information demandée.
- Ajouter aux boutons d'option des pays un second jeu de boutons d'option, par lesquels l'utilisateur choisit d'afficher les drapeaux dans un grand ou un petit format.

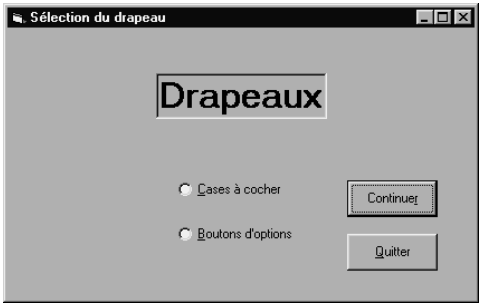
En plus de mettre en œuvre ces différents contrôles, l'application introduit un nouveau concept : plusieurs feuilles dans un même projet. En tout, trois feuilles distinctes seront créées. Vous apprendrez ainsi à charger et à afficher une feuille spécifique à un moment donné de l'exécution.

Comme ce sera le cas de beaucoup de programmes dans cet ouvrage, cette application utilise les fichiers graphiques livrés avec Visual Basic. Selon vos options d'installation, vous ne trouverez peut-être pas le sous-dossier Graphics dans le dossier Visual Basic de votre disque. Si c'est le cas, il faudra modifier les chemins d'accès utilisés dans l'application et travailler directement sur le premier CD-ROM de Visual Basic. Pour copier les fichiers graphiques sur votre disque, insérez ce même CD-ROM et choisissez Ajouter/Modifier des options.

Création de la première feuille

La Figure PB3.1 montre la première feuille que vous allez créer.

Figure PB3.1
Dans cette feuille, l'utilisateur choisit le type d'affichage des drapeaux.



Le Tableau PB3.1 détaille les propriétés des éléments de la feuille.

Tableau PB3.1 : Propriétés des contrôles de la première feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Caption`	`Sélection du drapeau`
Feuille	`Name`	`frmSelect`
Feuille	`Height`	`4035`
Feuille	`Width`	`6390`
Label	`Name`	`lblFlags`

Tableau PB3.1 : Propriétés des contrôles de la première feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Label	BorderStyle	1-Fixed Single
Label	Caption	Drapeaux
Label	Font	MS Sans Serif
Label	Font Size	24
Label	Font Style	Bold
Label	Height	615
Label	Left	2000
Label	Top	600
Label	Width	2300
Bouton d'option 1	Name	optCheck
Bouton d'option 1	Caption	&Cases à cocher
Bouton d'option 1	Left	2280
Bouton d'option 1	Top	1920
Bouton d'option 1	Width	1575
Bouton d'option 2	Name	optOption
Bouton d'option 2	Caption	&Boutons d'options
Bouton d'option 2	Left	2280
Bouton d'option 2	Top	2520
Bouton d'option 2	Width	1695
Bouton de commande 1	Name	cmdSelect
Bouton de commande 1	Caption	Continue&r
Bouton de commande 1	Left	4560
Bouton de commande 1	Top	2040

Tableau PB3.1 : Propriétés des contrôles de la première feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Bouton de commande 2	`Name`	`cmdExit`
Bouton de commande 2	`Caption`	`&Quitter`
Bouton de commande 2	`Left`	`4560`
Bouton de commande 2	`Top`	`2760`

Le code requis pour cette feuille est simple, mais il met en œuvre un nouveau concept. Ce code, donné au Listing PB3.1, implique le chargement d'une feuille distincte de celle qui est affichée.

Listing PB3.1 : Réception du choix de l'utilisateur par les boutons d'option

```
1:  Private Sub cmdSelect_Click()
2:  ' Vérifier l'absence d'erreur puis afficher
3:  ' la feuille selon les choix de l'utilisateur.
4:    Dim strMsg As String    ' Valeur renvoyée par la boîte de message
5:    If ((optCheck.Value = False) And (optOption.Value = False)) Then
6:      strMsg = MsgBox("Vous devez sélectionner une option",
        ➥vbCritical, "Erreur !")
7:    ElseIf (optCheck.Value = True) Then
8:           frmFlagsCheck.Show   ' Option cases à cocher.
9:       Else
10:          frmFlagsOpt.Show   ' Option boutons d'option.
11:   End If
12: End Sub
13:
14: Private Sub Form_Load()
15: ' Désélectionner tous les boutons d'option.
16:   optCheck.Value = False
17:   optOption.Value = False
18: End Sub
19: Private Sub cmdExit_Click()
20: ' fermer le programme.
21:   End
22: End Sub
```

Analyse de la première feuille

Les lignes 14 à 18 déterminent ce qui se passe lorsque l'application démarre et que la première feuille se charge. (La boîte de dialogue obtenue par le menu Projet, Propriétés devrait indiquer la feuille `frmSelect` comme objet de démarrage.) Les lignes 16 et 17 définissent les boutons d'option comme `False`, ce qui oblige l'utilisateur à en sélectionner un.

L'instruction conditionnelle de la ligne 5 détermine si l'utilisateur a cliqué sur le bouton de commande `Continuer` sans avoir sélectionné de bouton d'option. Si la propriété `Value` des deux boutons d'option est encore `False` lorsque l'utilisateur clique sur `Continuer`, la ligne 6 prend le relais et affiche un message d'erreur.

Si l'utilisateur a bien sélectionné l'un des boutons d'option, les lignes 7 à 9 déterminent de quel bouton il s'agit, et affichent la feuille correspondante. Notez le libellé de l'instruction utilisée : `frmFlagsCheck.Show`. Plutôt qu'à une commande, cela ferait plutôt penser à une sorte de valeur de propriété qui s'appellerait `Show`. Toutefois, il n'existe aucune valeur de propriété `Show` pour les feuilles. `Show` est, en fait, une *méthode*, c'est-à-dire non pas une commande Visual Basic classique (comme `Next`), mais une commande applicable uniquement à un objet spécifique. En l'occurrence, cet objet est la feuille `frmFlagsCheck`. La méthode `Show` affiche toute feuille à laquelle elle est appliquée. Résultat : dès que les lignes 8 ou 10 s'exécutent, la feuille correspondante s'affiche en haut de la feuille de sélection.

Création de la deuxième feuille

La Figure PB3.2 montre la feuille que nous allons créer.

La feuille affiche les noms de six pays, et six drapeaux peuvent apparaître à côté des noms. Il va falloir créer une nouvelle feuille et l'ajouter au projet courant. Voici comment vous allez procéder :

1. Cliquez du bouton droit dans la fenêtre Projet.
2. Dans le menu contextuel, sélectionnez Ajouter, puis Feuille. Visual Basic affiche une boîte de dialogue à onglets, dans laquelle vous pouvez générer une nouvelle feuille ou choisir une feuille existante.
3. Double-cliquez sur l'icône Form pour générer une nouvelle feuille. La feuille s'affiche dans la zone d'édition de la fenêtre Feuilles.

Le Tableau PB3.2 détaille les propriétés des éléments de la feuille. Rappelez-vous que l'application fait appel aux fichiers graphiques et que, s'ils n'ont pas été installés, il faudra le faire ou modifier les chemins d'accès affectés aux propriétés `Picture`.

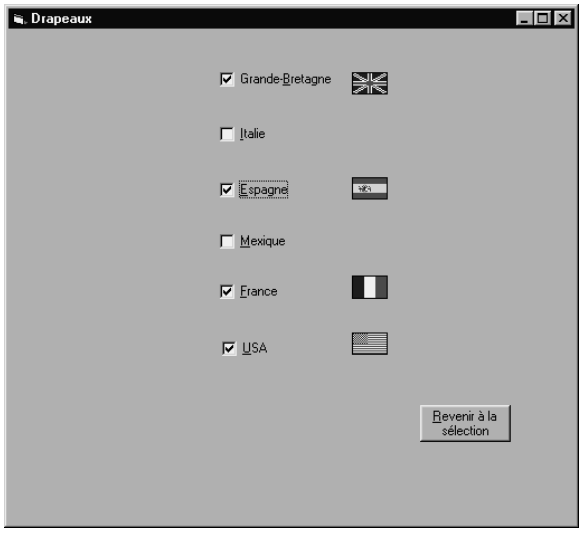


Figure PB3.2
L'utilisateur sélectionne le pays dont il veut afficher le drapeau.

Tableau PB3.2 : Propriétés des contrôles de la deuxième feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmFlagsCheck`
Feuille	`Caption`	`Drapeaux`
Feuille	`Height`	`7035`
Feuille	`Width`	`7710`
Case à cocher 1	`Name`	`chkEngland`
Case à cocher 1	`Caption`	`Grande-&Bretagne`
Case à cocher 1	`Left`	`2835`
Case à cocher 1	`Top`	`420`
Case à cocher 2	`Name`	`chkItaly`
Case à cocher 2	`Caption`	`&Italie`
Case à cocher 2	`Height`	`495`
Case à cocher 2	`Left`	`2835`

Tableau PB3.2 : Propriétés des contrôles de la deuxième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Case à cocher 2	`Top`	`1155`
Case à cocher 2	`Width`	`1215`
Case à cocher 3	`Name`	`chkSpain`
Case à cocher 3	`Caption`	`&Espagne`
Case à cocher 3	`Height`	`495`
Case à cocher 3	`Left`	`2835`
Case à cocher 3	`Top`	`1905`
Case à cocher 3	`Width`	`1215`
Case à cocher 4	`Name`	`chkMexico`
Case à cocher 4	`Caption`	`&Mexique`
Case à cocher 4	`Height`	`495`
Case à cocher 4	`Left`	`2835`
Case à cocher 4	`Top`	`2595`
Case à cocher 4	`Width`	`1215`
Case à cocher 5	`Name`	`chkFrance`
Case à cocher 5	`Caption`	`&France`
Case à cocher 5	`Height`	`495`
Case à cocher 5	`Left`	`2835`
Case à cocher 5	`Top`	`3375`
Case à cocher 5	`Width`	`1215`
Case à cocher 7	`Name`	`chkUSA`
Case à cocher 7	`Caption`	`&USA`
Case à cocher 7	`Height`	`495`

Tableau PB3.2 : Propriétés des contrôles de la deuxième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Case à cocher 7	`Left`	`2865`
Case à cocher 7	`Top`	`4140`
Case à cocher 7	`Width`	`1215`
Image 1	`Name`	`imgEngland`
Image 1	`Height`	`480`
Image 1	`Left`	`4440`
Image 1	`Picture`	`\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flaguk`
Image 1	`Top`	`480`
Image 1	`Visible`	`False`
Image 2	`Name`	`imgItaly`
Image 2	`Height`	`480`
Image 2	`Left`	`4440`
Image 2	`Picture`	`\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgitaly`
Image 2	`Top`	`1155`
Image 2	`Visible`	`False`
Image 3	`Name`	`imgSpain`
Image 3	`Height`	`480`
Image 3	`Left`	`4440`
Image 3	`Picture`	`\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgspain`
Image 3	`Top`	`1890`
Image 3	`Visible`	`False`
Image 4	`Name`	`imgMexico`

Tableau PB3.2 : Propriétés des contrôles de la deuxième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Image 4	`Height`	`480`
Image 4	`Left`	`4440`
Image 4	`Picture`	`\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgmex`
Image 4	`Top`	`2520`
Image 4	`Visible`	`False`
Image 5	`Name`	`imgFrance`
Image 5	`Height`	`480`
Image 5	`Left`	`4440`
Image 5	`Picture`	`\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgfran`
Image 5	`Top`	`3315`
Image 5	`Visible`	`False`
Image 6	`Name`	`imgUSA`
Image 6	`Height`	`480`
Image 6	`Left`	`4440`
Image 6	`Picture`	`\Program Files\Microsoft Visual Studio\Common\Graphics\Icons\Flags\Flgusa02`
Image 6	`Top`	`4080`
Image 6	`Visible`	`False`
Bouton de commande	`Name`	`cmdReturn`
Bouton de commande	`Caption`	`&Revenir à la sélection`
Bouton de commande	`Left`	`5520`
Bouton de commande	`Top`	`5040`

Il faut maintenant ajouter le code à la feuille. Double-cliquez sur la feuille `frmFlags-Check` et intégrez le code du Listing PB3.2. Cette nouvelle feuille devra afficher le drapeau correspondant à chaque pays coché par l'utilisateur. Il s'agit donc d'associer à chaque case à cocher une procédure événementielle `Click`.

Listing PB3.2 : Affichage des drapeaux pour chaque case cochée

```
1:  Private Sub chkEngland_Click()
2:    ' Case cochée = drapeau affiché.
3:    If chkEngland.Value = 1 Then
4:      imgEngland.Visible = True
5:    Else
6:      imgEngland.Visible = False
7:    End If
8:  End Sub
9:  Private Sub chkItaly_Click()
10:   ' Case cochée = drapeau affiché.
11:   If chkItaly.Value = 1 Then
12:     imgItaly.Visible = True
13:   Else
14:     imgItaly.Visible = False
15:   End If
16: End Sub
17: Private Sub chkSpain_Click()
18:   ' Case cochée = drapeau affiché.
19:   If chkSpain.Value = 1 Then
20:     imgSpain.Visible = True
21:   Else
22:     imgSpain.Visible = False
23:   End If
24: End Sub
25: Private Sub chkMexico_Click()
26:   ' Case cochée = drapeau affiché.
27:   If chkMexico.Value = 1 Then
28:     imgMexico.Visible = True
29:   Else
30:     imgMexico.Visible = False
31:   End If
32: End Sub
33: Private Sub chkFrance_Click()
34:   ' Case cochée = drapeau affiché.
35:   If chkFrance.Value = 1 Then
36:     imgFrance.Visible = True
37:   Else
38:     imgFrance.Visible = False
39:   End If
40: End Sub
41: Private Sub chkUSA_Click()
42:   ' Case cochée = drapeau affiché.
43:   If chkUSA.Value = 1 Then
```

```
44:     imgUSA.Visible = True
45:   Else
46:     imgUSA.Visible = False
47:   End If
48: End Sub
49: Private Sub cmdReturn_Click()
50: ' Retour à la feuille de sélection.
51:   frmFlagsCheck.Hide
52:   frmSelect.Show
53: End Sub
```

Analyse de la deuxième feuille

Les procédures événementielles des six cases à cocher sont identiques. Des chemins d'accès ont déjà été affectés à la propriété `Picture` de chaque contrôle image, lors de la création de la feuille. Pour afficher l'image, la procédure événementielle n'a donc plus qu'à définir comme `True` les diverses propriétés `Visible`. Reste un problème : que se passera-t-il si l'utilisateur clique de nouveau sur une case, désélectionnant ainsi l'option ? Le code doit être en mesure de désactiver l'affichage de l'image correspondante.

Les lignes 3 à 7 illustrent bien le fonctionnement des procédures événementielles. La ligne 3 interroge la propriété `Value` de la case à cocher. Si la valeur de cette propriété est `1`, l'utilisateur a coché la case, et le code doit afficher l'image. Si la valeur est `0`, la case a été décochée, et le code doit masquer l'image (ce que fait la ligne 6).

Enfin, la procédure événementielle `Click` du bouton de commande effectue deux opérations, aux lignes 51 et 52. La ligne 51 contient une nouvelle méthode, `Hide`, qui cache la feuille à laquelle elle est appliquée. (`Hide` a donc l'effet inverse de `Show`.) La ligne 51 cache la feuille des cases à cocher, tandis que la 52 affiche de nouveau la feuille de démarrage.

Création de la troisième feuille

La Figure PB3.3 montre la feuille que nous allons créer. Le drapeau pourra être grand ou petit, selon l'option sélectionnée par l'utilisateur.

La feuille présente un bouton d'option pour chacun des six noms de pays, et le drapeau du pays sélectionné s'affiche à côté du nom. Enfin, trois boutons d'option regroupés dans un frame permettent de définir la taille du drapeau.

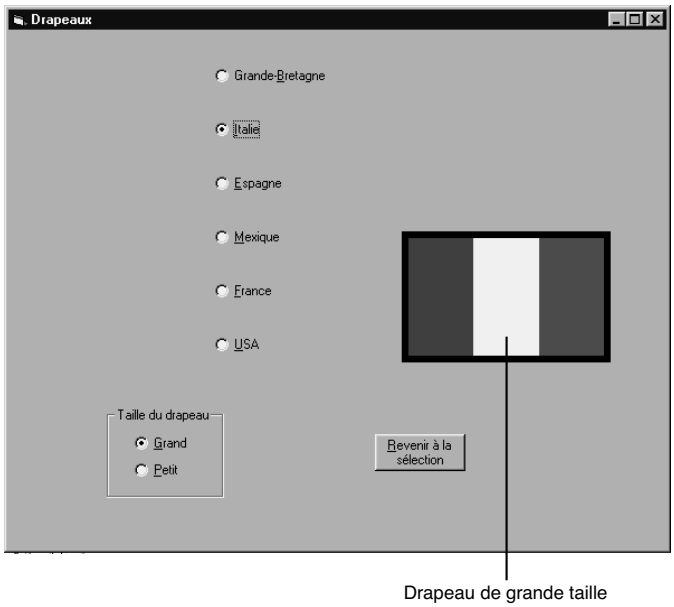


Figure PB3.3
Cette feuille permet à l'utilisateur d'afficher un drapeau à la fois et de définir la taille du drapeau.

Pour ajouter les boutons d'option au frame, il faut dessiner les boutons par-dessus le frame. Si vous vous contentez de double-cliquer sur l'outil OptionButton de la Boîte à outils, le bouton d'option apparaîtra au centre de la feuille, et Visual Basic ne le considérera pas comme une partie du frame. Pour que vos boutons d'option soient bien circonscrits au frame, il faut commencer par cliquer une fois sur l'outil, puis dessiner le contour du bouton sur le frame ; et ainsi, pour chaque bouton à ajouter. C'est à cette seule condition que les boutons d'option seront reconnus comme intégrés au frame, et distincts des autres boutons d'option de la feuille.

De nouveau, il faut créer une nouvelle feuille et l'ajouter au projet en cours. Une fois que vous avez ajouté cette troisième et dernière feuille, intégrez les éléments détaillés au Tableau PB3.3.

Tableau PB3.3 : Propriétés des contrôles de la troisième feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmFlagsOpt`
Feuille	`Caption`	`Drapeaux`
Feuille	`Height`	`7335`
Feuille	`Width`	`8955`
Bouton d'option 1	`Name`	`optEngland`
Bouton d'option 1	`Caption`	`Grande-&Bretagne`
Bouton d'option 1	`Height`	`495`
Bouton d'option 1	`Left`	`2760`
Bouton d'option 1	`Top`	`360`
Bouton d'option 1	`Value`	`True`
Bouton d'option 1	`Width`	`1215`
Bouton d'option 2	`Name`	`optItaly`
Bouton d'option 2	`Caption`	`&Italie`
Bouton d'option 2	`Height`	`495`
Bouton d'option 2	`Left`	`2760`
Bouton d'option 2	`Top`	`1080`
Bouton d'option 2	`Width`	`1215`
Bouton d'option 3	`Name`	`optSpain`
Bouton d'option 3	`Caption`	`&Espagne`
Bouton d'option 3	`Height`	`495`
Bouton d'option 3	`Left`	`2760`
Bouton d'option 3	`Top`	`1800`
Bouton d'option 3	`Width`	`1215`

Tableau PB3.3 : Propriétés des contrôles de la troisième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Bouton d'option 4	`Name`	`optMexico`
Bouton d'option 4	`Caption`	`&Mexique`
Bouton d'option 4	`Height`	`495`
Bouton d'option 4	`Left`	`2760`
Bouton d'option 4	`Top`	`2520`
Bouton d'option 4	`Width`	`1215`
Bouton d'option 5	`Name`	`optFrance`
Bouton d'option 5	`Caption`	`&France`
Bouton d'option 5	`Height`	`495`
Bouton d'option 5	`Left`	`2760`
Bouton d'option 5	`Top`	`3240`
Bouton d'option 5	`Width`	`1215`
Bouton d'option 6	`Name`	`optUSA`
Bouton d'option 6	`Caption`	`&USA`
Bouton d'option 6	`Height`	`495`
Bouton d'option 6	`Left`	`2760`
Bouton d'option 6	`Top`	`3960`
Bouton d'option 6	`Width`	`1215`
Frame	`Name`	`fraSize`
Frame	`Caption`	`Taille du drapeau`
Frame	`Height`	`1215`
Frame	`Left`	`1320`
Frame	`Top`	`5040`

Tableau PB3.3 : Propriétés des contrôles de la troisième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Frame	`Width`	`1575`
Frame, option 1	`Name`	`optLarge`
Frame, option 1	`Caption`	`&Grand`
Frame, option 1	`Height`	`255`
Frame, option 1	`Left`	`360`
Frame, option 1	`Top`	`360`
Frame, option 1	`Width`	`1095`
Frame, option 2	`Name`	`optSmall`
Frame, option 2	`Caption`	`&Petit`
Frame, option 2	`Height`	`255`
Frame, option 2	`Left`	`360`
Frame, option 2	`Top`	`720`
Frame, option 2	`Width`	`1095`
Image 1	`Name`	`imgEngland`
Image 1	`Height`	`480`
Image 1	`Left`	`5280`
Image 1	`Picture`	`\Program Files\Microsoft Visual Studio \Common\Graphics\Icons\Flags\Flguk`
Image 1	`Stretch`	`True`
Image 1	`Top`	`2160`
Image 1	`Visible`	`True`
Image 2	`Name`	`imgItaly`
Image 2	`Height`	`480`
Image 2	`Left`	`5280`

Tableau PB3.3 : Propriétés des contrôles de la troisième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Image 2	`Picture`	`\Program Files\Microsoft Visual Studio \Common\Graphics\Icons\Flags\Flgitaly`
Image 2	`Stretch`	`True`
Image 2	`Top`	`2160`
Image 2	`Visible`	`False`
Image 3	`Name`	`imgSpain`
Image 3	`Height`	`480`
Image 3	`Left`	`5280`
Image 3	`Picture`	`\Program Files\Microsoft Visual Studio \Common\Graphics\Icons\Flags\Flgspain`
Image 3	`Stretch`	`True`
Image 3	`Top`	`2160`
Image 3	`Visible`	`False`
Image 4	`Name`	`imgMexico`
Image 4	`Height`	`480`
Image 4	`Left`	`5280`
Image 4	`Picture`	`\Program Files\Microsoft Visual Studio \Common\Graphics\Icons\Flags\Flg\Flgmex`
Image 4	`Stretch`	`True`
Image 4	`Top`	`2160`
Image 4	`Visible`	`False`
Image 5	`Name`	`imgFrance`
Image 5	`Height`	`480`
Image 5	`Left`	`5280`
Image 5	`Picture`	`\Program Files\Microsoft Visual Studio \Common\Graphics\Icons\Flags\Flgfran`

Tableau PB3.3 : Propriétés des contrôles de la troisième feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Image 5	`Stretch`	`True`
Image 5	`Top`	`2160`
Image 5	`Visible`	`False`
Image 6	`Name`	`imgUSA`
Image 6	`Height`	`480`
Image 6	`Left`	`5280`
Image 6	`Picture`	`\Program Files\Microsoft Visua Studio \Common\Graphics\Icons\Flags\Flgusa02`
Image 6	`Stretch`	`True`
Image 6	`Top`	`2160`
Image 6	`Visible`	`False`
Bouton de commande	`Name`	`cmdReturn`
Bouton de commande	`Caption`	`&Revenir à la sélection`
Bouton de commande	`Height`	`495`
Bouton de commande	`Left`	`4920`
Bouton de commande	`Top`	`5400`
Bouton de commande	`Width`	`1215`

Le Listing PB3.3 fournit le code pour la troisième feuille. Comme vous pouvez le constater, il y a de la longueur ! Mais ce code, en réalité, consiste principalement en la répétition de six routines semblables, pour chacun des boutons d'option.

Listing PB3.3 : Affichage du drapeau selon le bouton d'option sélectionné

```
1:  Private Sub optEngland_Click()
2:     ' Bouton sélectionné = drapeau affiché.
3:     If optSmall.Value = True Then
4:        imgEngland.Height = 480
5:        imgEngland.Width = 480
6:     Else  ' Grande taille.
```

Listing PB3.3 : Affichage du drapeau selon le bouton d'option sélectionné *(suite)*

```
7:       imgEngland.Height = 2800
8:       imgEngland.Width = 2800
9:    End If
10:   imgEngland.Visible = True
11:    ' Masquer les autres drapeaux.
12:    imgItaly.Visible = False
13:    imgSpain.Visible = False
14:    imgMexico.Visible = False
15:    imgFrance.Visible = False
16:    imgUSA.Visible = False
17:  End Sub
18:  Private Sub optItaly_Click()
19:    ' Bouton sélectionné = drapeau affiché.
20:   If optSmall.Value = True Then
21:       imgItaly.Height = 480
22:       imgItaly.Width = 480
23:    Else  ' Grande taille.
24:       imgItaly.Height = 2800
25:       imgItaly.Width = 2800
26:    End If
27:    imgItaly.Visible = True
28:    ' Masquer les autres drapeaux.
29:    imgEngland.Visible = False
30:   imgSpain.Visible = False
31:    imgMexico.Visible = False
32:    imgFrance.Visible = False
33:    imgUSA.Visible = False
34:  End Sub
35:  Private Sub optSpain_Click()
36:  ' Bouton sélectionné = drapeau affiché.
37:    If optSmall.Value = True Then
38:       imgSpain.Height = 480
39:       imgSpain.Width = 480
40:   Else  ' Grande taille
41:       imgSpain.Height = 2800
42:       imgSpain.Width = 2800
43:    End If
44:    imgSpain.Visible = True
45:    ' Masquer les autres drapeaux.
46:    imgItaly.Visible = False
47:    imgEngland.Visible = False
48:    imgMexico.Visible = False
49:    imgFrance.Visible = False
50:   imgUSA.Visible = False
51:  End Sub
52:  Private Sub optMexico_Click()
53:    ' Bouton sélectionné = drapeau affiché.
54:    If optSmall.Value = True Then
55:       imgMexico.Height = 480
56:       imgMexico.Width = 480
```

```
57:    Else  ' Grande taille
58:       imgMexico.Height = 2800
59:       imgMexico.Width = 2800
60:   End If
61:    imgMexico.Visible = True
62:    ' Masquer les autres drapeaux.
63:    imgItaly.Visible = False
64:    imgSpain.Visible = False
65:    imgEngland.Visible = False
66:    imgFrance.Visible = False
67:    imgUSA.Visible = False
68:  End Sub
69:  Private Sub optFrance_Click()
70: ' Bouton sélectionné = drapeau affiché.
71:    If optSmall.Value = True Then
72:       imgFrance.Height = 480
73:       imgFrance.Width = 480
74:    Else  ' Grande taille
75:       imgFrance.Height = 2800
76:       imgFrance.Width = 2800
77:    End If
78:    imgFrance.Visible = True
79:    ' Masquer les autres drapeaux.
80:   imgItaly.Visible = False
81:    imgSpain.Visible = False
82:    imgMexico.Visible = False
83:    imgEngland.Visible = False
84:    imgUSA.Visible = False
85:  End Sub
86:  Private Sub optUSA_Click()
87:  ' Bouton sélectionné = drapeau affiché.
88:    If optSmall.Value = True Then
89:       imgUSA.Height = 480
90:      imgUSA.Width = 480
91:    Else  ' Grande taille
92:       imgUSA.Height = 2800
93:       imgUSA.Width = 2800
94:    End If
95:    imgUSA.Visible = True
96:    ' Masquer les autres drapeaux.
97:    imgItaly.Visible = False
98:    imgSpain.Visible = False
99:    imgMexico.Visible = False
100:   imgFrance.Visible = False
101:    imgEngland.Visible = False
102:  End Sub
103:  Private Sub cmdReturn_Click()
104:  ' Retour à la feuille de sélection.
105:    frmFlagsOpt.Hide
106:    frmSelect.Show
107:  End Sub
108:  Private Sub optSmall_Click()
```

Listing PB3.3 : Affichage du drapeau selon le bouton d'option sélectionné *(suite)*

```
109:  ' Masquer tous les drapeaux affichés.
110:  ' Les drapeaux seront maintenant petits.
111:     imgEngland.Visible = False
112:     imgItaly.Visible = False
113:     imgSpain.Visible = False
114:     imgMexico.Visible = False
115:     imgFrance.Visible = False
116:     imgUSA.Visible = False
117:     ' Désélectionner tous les boutons d'option
118:     optEngland.Value = False
119:     optItaly.Value = False
120:    optSpain.Value = False
121:     optMexico.Value = False
122:     optFrance.Value = False
123:     optUSA.Value = False
124:  End Sub
125:  Private Sub optLarge_Click()
126:  ' Masquer tous les drapeaux affichés.
127:  ' Les drapeaux seront maintenant petits.
128:     imgEngland.Visible = False
129:     imgItaly.Visible = False
130:    imgSpain.Visible = False
131:     imgMexico.Visible = False
132:     imgFrance.Visible = False
133:     imgUSA.Visible = False
134:     ' Désélectionner tous les boutons d'option
135:     optEngland.Value = False
136:     optItaly.Value = False
137:     optSpain.Value = False
138:     optMexico.Value = False
139:     optFrance.Value = False
140:    optUSA.Value = False
141: End Sub
```

Analyse de la troisième feuille

Le code est virtuellement identique pour la plupart des procédures événementielles, seul le nom du pays change. Examinez la première procédure, lignes 1 à 17. Le code vérifie d'abord quel bouton d'option a été sélectionné (ligne 3). Si c'est le bouton d'option "petite taille", les propriétés `Height` et `Width` du contrôle sont réglées à 480 twips (lignes 4 et 5). Si c'est le bouton d'option "grande taille" qui a été sélectionné, `Height` et `Width` sont réglées à 2800 twips (lignes 7 et 8).

Le reste de la procédure événementielle affiche simplement le drapeau du pays sélectionné, en définissant la propriété `Visible` comme `True` (ligne 10) et en désactivant

l'affichage des autres drapeaux (lignes 12 à 16). Les cinq procédures événementielles suivantes (lignes 18 à 102) font de même pour chaque drapeau sélectionnable.

A la ligne 103 commence une courte procédure événementielle qui fonctionne comme la procédure du module des cases à cocher. Elle cache la feuille des boutons d'option et affiche de nouveau la feuille de sélection, dans laquelle l'utilisateur pourra faire un autre choix ou quitter le programme.

Aux lignes 108 et 125 commencent des procédures événementielles identiques qui désactivent l'affichage de tous les drapeaux. Bien sûr, un seul drapeau était visible ; mais, au lieu de rechercher celui qui est affiché pour définir sa propriété `Visible` comme `False`, le code définit comme `False` la propriété `Visible` de tous les contrôles image. Même si cinq sur six ont déjà la valeur `False`, on s'assure ainsi qu'aucun drapeau ne sera visible.

En cliquant sur l'un des boutons d'option qui règlent la taille des drapeaux, on cache donc tous les drapeaux affichés ; cela afin que le drapeau choisi apparaisse dans l'une ou l'autre taille, selon la sélection de l'utilisateur. On pourrait réécrire l'application de sorte que le code change directement la taille du drapeau affiché, mais ce projet deviendrait encore plus long — et vous en avez déjà assez comme ça !

Voilà une application bien assommante ! Le code est répétitif — la même section répétée deux fois pour chacun des six boutons d'option et cases à cocher. Avant le terme de cet apprentissage, vous saurez rationaliser les codes de ce genre. Quand plusieurs contrôles similaires apparaissent sur une feuille, et qu'ils ont tous un comportement à peu près identique, on peut faire appel aux groupes de contrôles, qui raccourcissent le code et en facilitent la maintenance.

Résumé de la Partie I

Vous êtes au terme de la première partie. Vous pouvez être fier, car vous êtes en passe de devenir un authentique programmeur Visual Basic.

Comme vous l'avez compris, la programmation en Visual Basic implique bien plus que la simple maîtrise du langage. D'une certaine manière, le langage de programmation lui-même est secondaire. Créer une application Windows en Visual Basic, c'est surtout disposer des contrôles sur la feuille, puis régler les propriétés qui en définiront le comportement.

Voici ce que nous avons étudié dans ces sept chapitres :

- **L'historique de Visual Basic.** Visual Basic descend du langage BASIC (Chapitre 1).
- **La maintenance du programme.** Un programme clair et concis sera plus facile à modifier et à déboguer (Chapitre 1).
- **La structure du programme.** Les contrôles réagissent aux événements selon les propriétés que vous avez définies (Chapitre 1).
- **L'assistant Création d'applications.** Visual Basic crée pour vous l'ossature du programme. Vous n'avez plus qu'à ajouter les détails (Chapitre 1).
- **L'environnement Visual Basic.** L'environnement Visual Basic inclut toutes les fenêtres et barres d'outils requises pour programmer les applications Windows (Chapitre 2).
- **La fenêtre Présentation des feuilles.** Vous pouvez régler la disposition des feuilles sur votre écran (Chapitre 2).
- **L'Aide en ligne Visual Basic.** Visual Basic fournit toute l'aide en ligne nécessaire. Si vous avez installé MSDN, vous pouvez consulter, depuis Visual Basic, une immense source d'informations (Chapitre 2).

- **Créer des applications.** Plutôt que de passer par l'assistant Création d'applications, vous pouvez créer votre application à partir d'une feuille vierge et ajouter vous-même tout le code (Chapitre 2).
- **Les valeurs de propriétés.** Les propriétés des contrôles se définissent facilement, surtout dans la fenêtre Propriétés (Chapitre 2).
- **Le contrôle label.** Les labels permettent d'afficher du texte sur la feuille (Chapitre 3).
- **Le contrôle zone de texte.** Les zones de texte permettent à l'utilisateur de saisir des informations (Chapitre 3).
- **Le contrôle bouton de commande.** Les boutons de commande permettent à l'utilisateur de déclencher lui-même certaines actions (Chapitre 3).
- **Les menus.** Visual Basic vous permet d'intégrer à vos applications les barres et options de menu standards de Windows (Chapitre 4).
- **Le Créateur de menus.** Le Créateur de menus est une boîte de dialogue qui permet de créer rapidement un système de menus (Chapitre 4).
- **Les événements de menu.** Les options de menu déclenchent des événements `Click` qui sont faciles à programmer (Chapitre 4).
- **La fenêtre Code.** Les outils d'édition de la fenêtre Code vous assistent dans vos tâches d'écriture (Chapitre 5).
- **Les données.** Visual Basic supporte plusieurs types de données : nombres, chaînes, etc. (Chapitre 5).
- **Les variables.** Les variables permettent de stocker des valeurs et des résultats (Chapitre 5).
- **Les opérateurs.** Les opérateurs effectuent des opérations mathématiques et des traitements de données (Chapitre 5).
- **L'ordre de préséance des opérateurs.** Visual Basic calcule les expressions mathématiques selon un ordre strict de priorités (Chapitre 5).
- **Les opérateurs conditionnels.** En analysant des résultats, les opérateurs conditionnels permettent au programme de prendre des décisions (Chapitre 6).
- **Les instructions conditionnelles.** Les instructions `If` et `Select Case` se servent des opérateurs conditionnels pour orienter l'exécution du programme (Chapitre 6).
- **Les boucles.** Les boucles permettent de répéter plusieurs fois une section de code (Chapitre 6).

- **La gestion du clavier.** Des événements spécifiques permettent au programme de traiter le clavier de manière adéquate(Chapitre 7).
- **Le contrôle case à cocher.** Les cases à cocher offent à l'utilisateur un moyen simple de sélectionner des options (Chapitre 7).
- **Le contrôle bouton d'option.** Les boutons d'option permettent également de faire des sélections, mais mutuellement exclusives (Chapitre 7).
- **Le contrôle frame.** Les frames permettent de regrouper des boutons d'option qui forment dès lors un ensemble distinct des autres boutons d'option de la feuille (Chapitre 7).

Partie II

D'un coup d'œil

Comme vous avez pu le constater dans la Partie I, la programmation en Visual Basic est quelque chose de facile et même d'amusant parfois. Alors que les programmeurs d'avant Visual Basic devaient définir et configurer "à la main" les moindres éléments du programme, vous pouvez laisser à Visual Basic ces détails et vous concentrer sur les impératifs spécifiques de votre application. Dans cette Partie II, vous avancerez un peu plus vers la maîtrise de Visual Basic et travaillerez donc sur des applications un peu plus complexes.

Au terme des chapitres qui suivent, vous maîtriserez à peu près tout ce que vous devrez savoir du langage de programmation Visual Basic. Au Chapitre 9 notamment, vous découvrirez comment les nombreuses fonctions internes de Visual Basic vous épargnent une partie du codage. Vous pouvez ainsi remettre à Visual Basic le soin d'effectuer certains calculs et certaines manipulations.

Toutefois, cette partie ne parlera pas que de code. Vous apprendrez à intégrer à vos programmes des boîtes de dialogue standards, afin d'offrir à vos utilisateurs une interface familière pour l'ouverture de fichiers ou l'impression. Vos applications n'en seront que plus attrayantes.

Dans la Partie I, vous avez appris à interagir avec l'utilisateur par le biais des contrôles affichés à l'écran. Cette Partie II vous fera découvrir de nouvelles techniques d'interaction : répondre aux touches du clavier, aux clics et double-clics, aux opérations de glisser-déposer... Vous serez ainsi en mesure de satisfaire à toutes les attentes et à toutes les intuitions de l'utilisateur.

Mais ce n'est pas tout de recevoir des données — encore faut-il en envoyer à l'utilisateur. Vous découvrirez donc les techniques d'écriture de fichiers et d'impression. Enfin, au Chapitre 14, vous apprendrez à intégrer image et multimédia à vos applications. Grâce aux contrôles, aux propriétés, aux événements et aux méthodes Visual Basic, vous créerez des applications pleinement interactives et dynamiques… sans vous tuer à l'ouvrage !

Chapitre 8

Sous-routines et fonctions

Ce chapitre explique la structure et le fonctionnement des applications Visual Basic complexes, contenant de multiples modules et procédures. Dans de telles applications extensives, les données doivent être partagées entre plusieurs procédures et modules ; il faut, pour cela, déclarer convenablement les variables, et écrire le code de telle sorte que ces variables soient accessibles à plusieurs procédures.

Vous allez donc écrire de nombreuses procédures, mais également faire appel aux fonctions internes pour analyser et traiter des chaînes, des nombres et d'autres types de données. Ce chapitre présente toutes les fonctions internes dont vous aurez besoin en situation réelle.

Voici ce que nous découvrirons aujourd'hui :

- la structure des programmes ;
- les procédures générales ;
- la portée des variables ;
- les listes d'arguments ;
- les fonctions numériques ;
- les fonctions d'analyse de données ;
- les fonctions de manipulation de chaînes ;
- les fonctions de date et d'heure.

Questions de structures

Vous avez déjà une bonne idée du fonctionnement interne des programmes Visual Basic. Lorsque l'utilisateur interagit avec les contrôles, des événements ont lieu. Si l'application contient les procédures événementielles adaptées à ces contrôles et à ces événements, le programme réagit. Le code d'un programme Visual Basic est, pour le principal, constitué d'une section de déclarations, suivie d'une longue série de procédures événementielles.

Rappelez-vous qu'à chaque feuille correspond un module de feuille qui contient le code de la feuille et de ses contrôles. Lorsqu'une feuille est affichée dans la zone d'édition de la fenêtre Feuilles, on affiche le code correspondant en cliquant sur le bouton Code de la fenêtre Projet.

Nous allons maintenant nous attacher à une autre partie des programmes Visual Basic. Les procédures événementielles ne constituent pas le seul type de procédures susceptible d'apparaître dans le code. La Figure 8.1 illustre le code contenu dans un module de feuille. Outre les déclarations et les procédures événementielles, un module peut contenir des *procédures générales* et des *procédures de classe*.

Figure 8.1
Un module recèle différents types de codes.

Le module de feuille

Déclarations
Private Sub Gen_Proc1 []□ ' Corps de la première procédure générale.□ End Sub
Private Sub Gen_Proc2 []□ ' Corps de la seconde procédure générale.□ End Sub
Private Sub Event_Proc1 []□ ' Corps de la première procédure événementielle.□ End Sub
Private Sub Event_Proc2 []□ ' Corps de la seconde procédure événementielle.□ End Suba
Private Sub Class_Proc1 []□ ' Corps de la première procédure de classe.□ End Sub

Une procédure est dite générale quand elle n'est pas associée à un événement de contrôle, mais effectue des calculs et divers traitements.

Une procédure de classe est un objet Visual Basic spécial, créé par le programmeur. Les classes permettent de définir un nouveau type de données ou de variables.

Souvenez-vous des deux listes déroulantes qui, dans la fenêtre Code, indiquent le type ou le nom des objets et des procédures du module en cours. La liste Objet affiche une entrée pour chaque contrôle de la feuille. Tout en haut de la liste se touve également un objet spécial, `(Général)`. (Les parenthèses indiquent qu'il ne s'agit pas d'un contrôle, mais d'une section du code.) La section générale contient à la fois la section de déclarations, située au début du code, et les procédures générales que vous écrivez.

Les appels de procédures générales

Les procédures générales peuvent être des fonctions ou des sous-routines. A quoi sert une procédure générale ? Dans le Projet bonus qui précédait ce chapitre, "Variables et expressions", deux procédures événementielles (`optLarge _Click()` et `optSmall_Click()`) apparaissaient, qui contenaient le code suivant :

```
imgEngland.Visible = False
imgItaly.Visible = False
imgSpain.Visible = False
imgMexico.Visible = False
imgFrance.Visible = False
imgUSA.Visible = False
' Désélectionner tous les boutons d'option.
optEngland.Value = False
optItaly.Value = False
optSpain.Value = False
optMexico.Value = False
optFrance.Value = False
optUSA.Value = False
```

Que de lignes pour dire la même chose ! Bien entendu, la fenêtre Code autorise le copier/coller, ce qui peut alléger votre peine, mais ne résout pas le problème. Que se passera-t-il si vous devez modifier le code ? Il faudra alors avoir soin de mettre à jour chaque ligne concernée. Le code répétitif n'est pas seulement fatigant à saisir, mais également plus délicat à maintenir.

Dans des cas pareils, la meilleure solution consiste à placer le code dans sa propre procédure générale, comme dans le code suivant. Notez que chaque procédure générale doit recevoir un nom unique (suivez pour cela les mêmes règles que pour les noms de variables).

En outre, une procédure générale peut être une fonction aussi bien qu'une sous-routine. Examinez cela :

```
1:  Private Sub Clear_Flags()
2:  ' Masquer tous les drapeaux affichés.
3:  ' Les drapeaux seront maintenant petits.
4:     imgEngland.Visible = False
5:     imgItaly.Visible = False
6:     imgSpain.Visible = False
7:     imgMexico.Visible = False
8:     imgFrance.Visible = False
9:     imgUSA.Visible = False
10:    ' Désélectionner tous les boutons d'option.
11:     optEngland.Value = False
12:     optItaly.Value = False
13:     optSpain.Value = False
14:     optMexico.Value = False
15:     optFrance.Value = False
16:     optUSA.Value = False
17:  End Sub
```

Lorsqu'une autre procédure requiert l'exécution d'un code identique, il suffit d'*appeler* la procédure générale, comme ceci :

```
Call Clear_Flags()
```

Appeler une procédure signifie exécuter une procédure depuis une autre procédure.

L'instruction `Call` indique à Visual Basic de mettre temporairement de côté la procédure en cours d'exécution (qu'il s'agisse d'une procédure événementielle ou d'une procédure générale), et d'exécuter le code de la procédure appelée. Une fois que la procédure appelée est entièrement exécutée, l'exécution de la procédure appelante se poursuit à partir de la ligne qui suit l'instruction `Call`.

Procédures privées et publiques

L'instruction `Call` vous économise de la peine et du temps, et elle facilite considérablement la maintenance : le code commun est stocké une fois pour toutes dans une procédure générale, qui pourra être appelée dans la suite du programme à l'aide d'une simple instruction. Mais vous pouvez aussi écrire une routine dans une application et la rendre disponible aux autres applications. Exemple : dans une application qui génère des rapports, vous créez un en-tête indiquant le nom et l'adresse de la société ; cet en-tête pourra être

inclus dans d'autres rapports générés par d'autres applications. Comment ? C'est ce que nous allons maintenant vous expliquer.

Une procédure située dans la section générale d'un module de feuille est disponible pour tout le module, mais seulement pour ce module ; les autres applications n'y ont pas accès. Pour cela, il faut placer la procédure dans un module de code. Vous pouvez ainsi écrire le code de vos rapports dans un module de code spécifique qui contiendra l'ensemble des routines utilisées pour générer des rapports. Grâce au module de code, ces routines seront alors accessibles à toute application impliquant la génération de rapports. Il suffit de cliquer du bouton droit dans la fenêtre Projet de l'application, de sélectionner dans le menu contextuel Ajouter, Module, puis de saisir le code.

On pourrait dire que, en écrivant des procédures générales accessibles à toutes les applications, vous construisez votre propre bibliothèque de fonctions internes. Il ne s'agit pas vraiment de fonctions internes (ou, plus précisément, de fonctions intrinsèques), puisqu'elles ne font pas partie du système Visual Basic ; mais ce sont des outils que vous forgez vous-même pour vous-même, et qui pourront servir dans chacun de vos projets.

Le code d'un module de feuille peut exploiter un module de code ajouté. Il convient, pour cela, d'appeler la procédure depuis le module de feuille. Une exception toutefois : seules les procédures *publiques* peuvent être utilisées en dehors de leur propre module, contrairement aux procédures *privées*. Considérez cette déclaration de procédure :

```
Private Sub ReportIt()
```

Cette procédure peut seulement être appelée dans le module qui la contient. Si vous l'aviez déclarée comme procédure publique, elle aurait été accessible depuis n'importe quelle autre procédure de n'importe quel autre module de l'application. Voici une déclaration de procédure publique :

```
Public Sub ReportIt()
```

Ainsi, il convient de déclarer comme publiques les procédures d'utilité générale que vous écrivez afin de disposer toujours de son code.

Récapitulons :

- Une procédure déclarée comme `Private` n'est exploitable qu'à l'intérieur de son propre module.
- Une procédure déclarée comme `Public` peut être appelée depuis n'importe quelle autre procédure de l'application.

La portée des variables

Si le code est lui-même tantôt public, tantôt privé, les variables peuvent également avoir une *portée* publique ou privée ; le jargon parle plus volontiers de *variables globales* ou *locales*. Les variables sont ainsi distinguées selon qu'elles sont ou non accessibles pour un autre code.

Une variable locale n'est exploitable que dans le code qui la contient.

Une variable globale est exploitable en dehors de la zone pour laquelle elle est déclarée.

On entend par portée le degré de disponibilité d'une variable dans le code d'une application.

Les contrôles sont tous et toujours visibles et publics pour l'ensemble du code d'une application. Les contrôles d'une feuille ne sont jamais cachés.

Imaginons qu'un commerçant vous charge d'écrire une petite application comptable, destinée notamment à calculer le montant d'une taxe de vente. Vous écrivez donc une procédure qui calcule la taxe sur le chiffre de vente total. Mais cette procédure est, en fait, d'utilité générale, puisque la taxe s'applique à différents types de produits, traités dans des modules différents. Plutôt que de répéter les routines de calcul plusieurs fois dans l'application, vous les stockez dans un module de code spécifique. La procédure de calcul sera ainsi accessible à toutes les applications qui doivent prendre en compte le calcul de cette taxe de vente ; il suffira que vous leur ajoutiez le module.

Reste toutefois un problème — les procédures ne peuvent pas toujours partager les données. Considérez la section de code suivante :

```
Private Sub GetSalesTotal()
' Ajouter le prix de chaque article vendu,
' puis calculer le chiffre de vente total.
  Dim curTotal As Currency
  '
  ' Suite de la procédure.
```

La procédure `GetSalesTotal()` ajoute le contenu de toutes les zones de texte d'un formulaire de vente, et les stocke dans la variable `curTotal`. Il faudrait ensuite faire appel à la procédure externe pour calculer le montant de la taxe de vente sur la somme `curTotal`. Eh bien, ça ne marchera pas ! Pourquoi ? Parce que `curTotal` est une variable *locale*.

Nous avons vu qu'une variable locale n'est utilisable que pour le code de la procédure dans laquelle elle a été déclarée. On dit aussi que la variable n'est visible que pour cette procédure.

Le terme visible renvoie au concept de portée expliqué plus haut. Une variable est visible pour les procédures qui y ont accès. Ainsi, une variable locale n'est visible que pour la procédure qui la contient, tandis qu'une variable globale est visible pour tout le module.

Vous pouvez déclarer `curTotal` comme variable globale pour le module, en déplaçant l'instruction `Dim` de la procédure à la section de déclarations du module :

```
Option Explicit
' Déclaration de toutes les variables publiques.
Dim curTotal As Currency
```

La variable `curTotal` est alors accessible partout dans le module de feuille. En fait, cela va sans doute vous surprendre, mais la situation est encore *pire*. Comme nous l'expliquions au Chapitre 5, les variables locales sont préférables aux variables globales, parce que, d'après une règle générale, les procédures ne doivent avoir accès qu'aux données strictement nécessaires. En la déclarant comme globale dans la section `(Général)` de la fenêtre Code, vous avez rendu la variable visible pour toutes les procédures du module, même à celles qui n'en ont aucun usage ; mais la variable n'est toujours pas visible pour les autres modules !

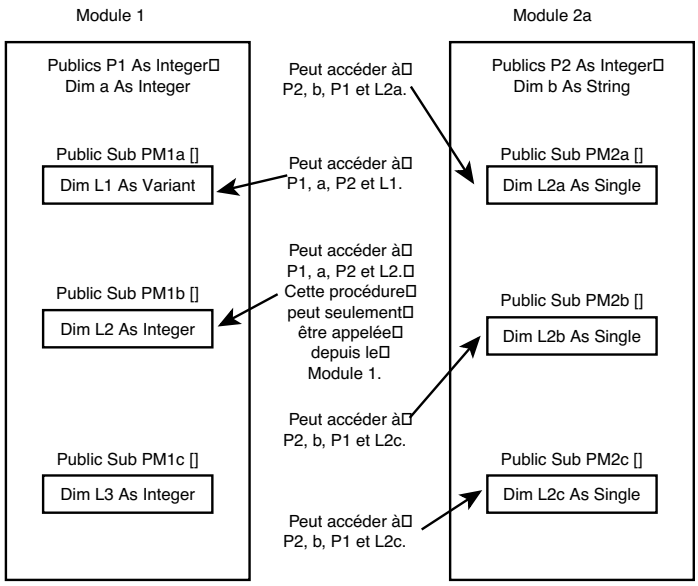
Les applications Visual Basic contenant en général plusieurs modules, les termes locale et globale ne font pas sens dans toutes les situations. Il serait plus convenable de désigner comme locale une variable visible pour sa procédure (celle qui contient la déclaration) ; de désigner comme variable de niveau module une variable visible pour son module ; et de désigner comme globale ou publique une variable visible pour toute l'application.

Pour que la variable soit réellement globale à tout le projet, il faut la déclarer avec le mot clé `Public`, au lieu de l'usuel `Dim`, et placer cette instruction `Public` dans la section de déclarations du module. La variable devient ainsi publique pour l'ensemble de l'application.

Deux variables publiques peuvent porter le même nom, à condition d'être déclarées comme `Public` dans deux modules séparés. Afin de ne jamais vous mélanger les pinceaux, il convient d'inclure dans le nom des variables publiques le nom de leur module d'origine. Ainsi, `MaBiblio.curSales` renvoie spécifiquement à la variable `curSales` du module de code `MaBiblio`. Même si une autre variable publique `curSales` existe dans un autre module, Visual Basic ne les confondra jamais.

A titre de récapitulatif des notions étudiées jusqu'ici, examinez la Figure 8.2. Y sont illustrées les relations qu'entretiennent dans une application les variables publiques, de niveau module et locales. Dans chaque module, les rectangles représentent des procédures. Les annotations indiquent quelles variables sont accessibles à quelles procédures.

Figure 8.2
La portée d'une variable détermine les procédures qui peuvent y accéder.



Toutes ces histoires de variables publiques ont de quoi désoler les programmeurs soucieux de lisibilité et de sûreté du code. Car, si les variables de niveau module sont à éviter, que dire, alors, des variables déclarées comme publiques pour une application — c'est-à-dire visibles pour toutes les procédures de tous les modules du projet !

En définitive, et sauf les rares cas où une même variable doit être utilisée par presque toutes les procédures d'un module, il est préférable de déclarer des variables locales avec `Dim`. Mais d'autres procédures peuvent, malgré tout, avoir besoin de ces variables, notamment les procédures générales stockées dans des modules de code externes. Il doit donc exister un moyen de partager des données locales. C'est ce que dévoile la prochaine section.

La transmission des données

Lorsque deux procédures doivent partager des données, la procédure appelante peut envoyer ses variables locales à la procédure appelée. Pour ainsi transmettre des variables, il faut les indiquer dans les parenthèses de la procédure appelée. Vous avez déjà eu l'occasion de transmettre des arguments à une procédure, par exemple avec `InputBox()`. La chaîne spécifiée entre les parenthèses de `InputBox()` était une donnée locale pour la procédure qui appelait `InputBox()`, et `InputBox()` recevait et traitait cette chaîne.

La règle consiste donc à déclarer dans les parenthèses de la procédure appelée tous les arguments qui lui sont passés, transmis. Un simple exemple devrait vous éclairer. Le Listing 8.1 contient deux procédures. La première envoie deux valeurs, un total et un pourcentage d'abattement fiscal, à la seconde, qui calcule une taxe de vente sur le total moins l'abattement. La procédure de calcul de la taxe traite les données qui lui sont transmises par la première procédure, et affiche la taxation totale dans une boîte de message.

Listing 8.1 : La première procédure passe des arguments à la seconde

```
1:  Private Sub GetTotal()
2:  ' Cette procédure additionne les valeurs d'un
3:  ' formulaire, puis envoie le total et le pourcentage
4:  ' d'abattement à la procédure qui calcule la taxe.
5:     Dim curTotal As Currency
6:     Dim sngDisc As Single   ' Special tax discount
7:     '
8:     ' Calculer le total d'après le formulaire.
9:     curTotal = txtSale1.Text + txtSale2.Text + txtSale3.txt
10:    '
11:    ' Envoyer le total à la procédure de taxation.
12:    Call SalesTax(curTotal, sngDisc)
13: End Sub
14:
15: Public Sub SalesTax(curTotal As Currency, sngRateDisc As Single)
16: ' Calculer la taxe et déduire l'abattement.
17:   Dim curSalesTax As Currency
18:   Dim intMsg As Integer  ' For MsgBox()
19:   '
20:   ' Calcul de la taxe de vente
21:   ' à 3,5 % du total.
22:   curSalesTax = (curTotal * .035)
23:   '
24:   ' Déduction du pourcentage d'abattement.
25:   curSalesTax = curSalesTax - (sngRateDisc * curTotal)
26:   '
27:   ' Afficher le montant total de la taxation.
28:   intMsg = MsgBox("Taxation totale : " & curSalesTax)
29:   '
30:   ' Après exécution, les procédures redonnent
31:   ' la main à la procédure appelante.
32: End Sub
```

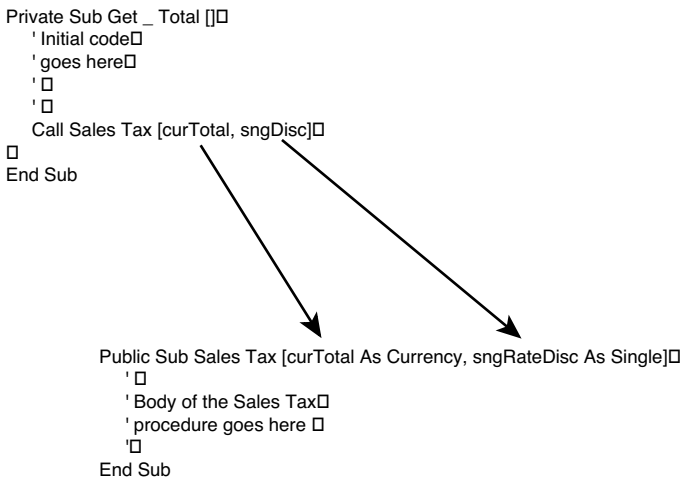
Notez que le nom des variables passées ne doit pas forcément correspondre au nom des arguments reçus (ainsi, `sngDisc` à la ligne 12 et `sngRateDisc` à la ligne 15). En revanche, le nombre d'arguments et les types de données doivent être identiques dans la liste d'arguments passés et dans la liste d'arguments reçus. La procédure appelée s'en tient aux noms sous lesquels les arguments lui ont été passés.

La procédure `SalesTax()` peut être stockée dans un module de code différent de celui de la procédure `GetTotal()`, laquelle peut résider dans la section générale d'un module de feuille. Les deux variables passées, bien que locales pour `GetTotal()`, sont passées comme arguments à la procédure externe `SalesTax()`. Ici, `SalesTax()` doit nécessairement recevoir des arguments au nombre de deux, et du type de données spécifié.

Une fois les deux variables reçues, la procédure `SalesTax()` peut les traiter comme s'il s'agissait de variables locales pour elle-même. Les calculs sont appliqués aux valeurs, et la ligne 28 affiche le résultat (le montant total de la taxe) dans une boîte de message.

La Figure 8.3 illustre la transmission des arguments de `GetTotal()` à `SalesTax()`.

Figure 8.3
GetTotal() envoie deux arguments à SalesTax().



La ligne semble utiliser des types de données non équivalents, mais Visual Basic sait gérer ce type de non-équivalence. `MsgBox()` requiert un argument initial de type `String`, et l'opérateur `&` ne peut concaténer que des chaînes. Mais lorsqu'on demande la concaténation d'une chaîne et d'un nombre, Visual Basic convertit automatiquement le nombre en chaîne.

Faire

Pour interrompre une sous-routine avant son instruction `End Sub`*, utilisez* `Exit Sub`*. (Pour les procédures de type fonction, ce sera* `End Function`*.) Toutefois,* `Exit Sub` *devrait toujours être incluse dans une instruction* `If` *(ou autre instruction conditionnelle), sans quoi la procédure s'arrêterait systématiquement et n'irait jamais plus loin que* `Exit Sub`*.*

Il y a une autre façon d'appeler les sous-routines. Vous pouvez omettre l'instruction `Call` ainsi que les parenthèses. Les deux instructions suivantes sont équivalentes.

```
Call PrintTitle (Title)
PrintTitle Title
```

Transmission par référence et par valeur

Dans le Listing 8.1, les variables sont passées selon la méthode dite *par référence* ; ce qui signifie que la procédure appelée peut modifier les arguments de la procédure appelante. Si, en revanche, les arguments sont précédés du mot clé `ByVal`, ils sont passés *par valeur*, c'est-à-dire que la procédure appelée ne peut pas modifier les arguments de la procédure appelante. (La transmission par référence étant la méthode par défaut, le mot clé `ByRef` est toujours optionnel.)

Prenons l'exemple du Listing 8.1. Si la procédure `SalesTax()` modifiait la valeur des arguments `curTotal` ou `sngRateDisc`, la variable correspondante dans `GetTotal()` serait également modifiée. Selon la méthode de transmission par référence, les arguments passés par la procédure appelante peuvent être affectés par les traitements qui leur sont appliqués dans la procédure appelée.

Si la ligne 15 du Listing 8.1 avait été écrite comme l'instruction ci-dessous, les variables de `GetTotal()` variables auraient été protégées contre toute modification par `SalesTax()`. Dès l'exécution de `SalesTax()` terminée, la valeur initiale des arguments passés par `GetTotal()` aurait été restaurée.

```
Public Sub SalesTax(ByVal curTotal As Currency, ByVal
➥sngRateDisc As Single)
```

Ainsi, et à moins que la procédure appelée soit expressément destinée à modifier les arguments passés, les arguments devraient toujours être protégés par le mot clé `ByVal`. De cette façon, la procédure appelée peut triturer tant qu'elle veut les arguments, la procédure appelante retrouvera toujours ses valeurs d'origine.

Le mot clé `ByVal` *peut être utilisé devant tous, une partie ou un seul des arguments passés.*

Les appels de fonctions

Les fonctions se distinguent des sous-routines non seulement pas les lignes d'encadrement, mais aussi par la façon de renvoyer les valeurs à la procédure appelante. Dans les sections précédentes, le code appelait des procédures de type sous-routine. L'appel d'une procédure de type fonction est légèrement différent. En fait, on appelle une procédure fonction comme on appelle une fonction interne — avec le nom de la fonction et avec ses arguments —, mais sans l'instruction `Call`. La fonction devient alors elle-même la valeur renvoyée, et cette valeur est utilisée dans une expression ou une dans une instruction.

La première ligne d'une procédure fonction, ou *déclaration de fonction*, doit suivre ce format :

```
Public|Private Function FuncName (Arg)As dataType
```

Ici, *`FuncName`* est le nom de la fonction, *`Arg`* la liste des arguments, et *`dataType`* le type de données. La barre verticale entre `Public` et `Private` indique qu'une fonction est soit privée (visible pour le module seulement), soit publique (visible pour l'application entière). Une fonction peut ne pas avoir d'arguments ; auquel cas, on omet les parenthèses. Toutefois, la plupart des fonctions reçoivent au moins un argument. La clause `As` *`dataType`*, que l'on ne trouve pas dans les procédures de type sous-routine, déclare le type de données de la valeur renvoyée par la fonction. Une fonction ne peut renvoyer qu'une seule valeur dont le type de données est spécifié par *`dataType`*.

Si l'on réécrivait la procédure `SalesTax()` en une fonction qui renvoie le résultat du calcul de la taxe, la déclaration serait la suivante :

```
Public Function SalesTax(curTotal As Currency, sngRateDisc
➥As Single) As Currency
```

Tout comme aux sous-routines, les arguments sont passés aux fonctions par référence ou par valeur, selon le degré de protection que vous souhaitez assurer aux arguments de la fonction appelante.

La valeur renvoyée doit nécessairement être affectée à une variable portant le même nom que la fonction. Un telle variable n'a pas à être déclarée. Ainsi, la valeur affectée par `SalesTax()` à une variable nommée `SalesTax()` serait la valeur renvoyée par la fonction. Lorsque l'exécution de la fonction s'achève, soit par `End Function`, soit par une instruction `Exit Function`, la valeur renvoyée par la fonction est stockée dans la variable de renvoi.

La procédure appelante doit donc réserver une place pour la valeur renvoyée. La valeur renvoyée est généralement affectée à une variable. Il n'est pas rare de trouver, dans une procédure appelante, une telle instruction :

```
curDailyNet = CalcNetAdj(curGrSls, sngTaxRt, curGrRate, curStrExp)
```

`CalcNetAdj()` est une fonction à laquelle quatre valeurs sont passées. `CalcNetAdj()` traite ces valeurs, et affecte une valeur de renvoi à la variable nommée `CalcNetAdj`. Au terme de la fonction, cette valeur est affectée à la variable `curDailyNet` de la procédure appelante.

Les exercices de fin de chapitre vous invitent à transformer en fonction la sous-routine `CalcTax()` du Listing 8.1. Pour le moment, étudiez le Listing 8.2 afin de bien assurer votre compréhension des fonctions.

Listing 8.2 : Les fonctions renvoient à la procédure appelante une valeur unique

```
1:  ' La procédure appelante commence ici.
2:  Private Sub CP()
3:     Dim varR As Variant    ' Variables locales à l'origine
4:     Dim varV As Variant    ' de la valeur renvoyée.
5:     Dim intI As Integer    ' Contiendra la valeur renvoyée.
6:
7:     varR = 32              ' Valeurs initiales.
8:     varV = 64
9:
10:    intI = RF(varR, varV)  ' Passe varR et varV.
11:                           ' intI reçoit la valeur renvoyée.
12:    MsgBox("Après renvoi, intI vaut " & intI)
13:    MsgBox("Après renvoi, varR vaut " & varR)
14:    MsgBox("Après renvoi, varV vaut " & varV)
15:
16: End Sub
17: ' La fonction appelée commence ici.
18: Public Function RF (varR As Variant, ByVal varV As Variant) As Integer
19: ' varR est reçu par référence et varV par valeur.
20:    varR = 81         ' Modifie les deux arguments.
21:    varV = varV + 10
22:    ' Définit la valeur de renvoi.
23:    RF = varR + varV
24: End Function
```

La ligne 10 passe `varR` et `varV` (32 et 64) à la fonction. Aux lignes 20 et 21, la fonction affecte à `varR` et `varV` les valeurs 81 et 74. `varR` ayant été passé par référence, sa valeur sera également de 81 dans la procédure appelante (`CP()`). La ligne 23 additionne les valeurs 81 et 74, et elle affecte la somme comme valeur de renvoi de la fonction. Lorsque `End Function` s'exécute, l'instruction d'affectation de la procédure appelante (ligne 10) affecte 155 à `intI`. La ligne 12 affiche la valeur de `intI`. La ligne 13 affiche la valeur 81, montrant ainsi que `varR` a été modifié par la fonction appelée. La ligne 14 affiche toujours 64, la valeur de `varV` ayant été protégée par le mot clé `ByVal`.

Transmission des contrôles comme arguments

Outre les variables, les contrôles peuvent eux aussi être passés d'une procédure à l'autre. Vous pouvez ainsi écrire une procédure qui traite la valeur d'un contrôle, mais il faut pour cela savoir quel type de contrôle a été passé. La procédure peut être appelée depuis différents modules, et différents contrôles sont passés selon le cas. Le jeu d'instructions `If TypeOf` permet de vérifier le type de données du contrôle passé comme argument.

Voici le format d'une instruction `If` classique incluant `TypeOf` :

```
If TypeOf object Is objectType Then
   Bloc d'instructions Visual Basic.
Else
   Bloc d'instructions Visual Basic.
End If
```

Ici, *`object`* peut être tout argument ou *variable de contrôle*. *`objectType`* est l'une des valeurs suivantes :

```
CheckBox, ComboBox, CommandButton, Graph, Image, Label, Line,
➥ListBox, OptionButton, OptionGroup, PageBreak, PictureBox,
➥Rectangle, Shape, TextBox, ToggleButton
```

Une variable de contrôle (on dit aussi variable d'objet) est une variable déclarée comme contrôle. Les variables, nous le savons, prennent les types de données `Integer`, `String` ou `Currency`. Mais les variables peuvent être déclarées comme `Object`, c'est-à-dire comme n'importe quel objet Visual Basic, contrôles inclus.

Les instructions suivantes déclarent des variables de contrôles :

```
Dim objCmdFirst As CommandButton
Dim objNameList As ListBox
Dim objPhoto As Image
```

Les variables de contrôle sont également transmissibles comme arguments :

```
Public Function FixControls (objIncoming As Object)
```

Quand une fonction reçoit un argument déclaré comme de type `Object`, on peut vérifier de quel contrôle il s'agit :

```
If TypeOf objIncoming Is CommandButton Then
   MsgBox("C'est un bouton de commande")
ElseIf TypeOf objIncoming Is CheckBox Then
```

```
   MsgBox("C'est une case à cocher")
ElseIf TypeOf objIncoming Is TextBox Then
   MsgBox("C'est une zone de texte")
End If
```

Les fonctions internes

Vous connaissez déjà les fonctions internes `LoadPicture()`, `MsgBox()` et `InputBox()`. Il en existe beaucoup, beaucoup d'autres… La suite de ce chapitre vous présente les plus importantes, celles qui vous aideront à construire des programmes plus puissants. Ainsi, au terme de ce chapitre, vous saurez non seulement écrire vos propres sous-routines et procédures, mais vous connaîtrez également la plupart des fonctions internes. Grâce à cette maîtrise du langage Visual Basic, vous pourrez, dès le prochain chapitre, créer des applications plus avancées et plus complexes qu'elles ne l'ont été jusque-là.

Lorsque vous aurez terminé le présent chapitre, vous connaîtrez l'essentiel du langage Visual Basic. S'il vous restera encore des commandes à découvrir, vous maîtriserez tous les éléments nécessaires au programmeur débutant et moyen. Ce qui ne signifie pas que le reste soit de la rigolade ! Mais nombre de chapitres suivants introduisent simplement de nouveaux contrôles et propriétés.

Fonctions numériques

Nous commencerons notre étude des fonctions internes par les fonctions de conversion en entier. Les plus communes ont ce format :

```
Int(numericValue)
Fix(numericValue)
```

Ici, *`numericValue`* est un littéral, une variable ou une expression numériques, et même une fonction imbriquée capable de renvoyer un nombre. Le type de données numérique passé, quel qu'il soit, est maintenu pour les valeurs renvoyées ; mais ces valeurs sont utilisables comme des entiers `Integer`.

Ne pas faire
Ne passez jamais à `Int()` ou `Fix()` un argument non numérique. Visual Basic générerait une erreur à l'exécution.

Ces deux fonctions ont pour effet d'arrondir leurs arguments à l'entier le plus proche. Elles se distinguent toutefois par le traitement appliqué aux valeurs négatives. Dans les

instructions suivantes, les commentaires indiquent la valeur renvoyée par chaque fonction :

```
intAns1 = Int(6.8)    ' 6
intAns2 = Fix(6.8)    ' 6
intAns3 = Int(-6.8)   ' -7
intAns4 = Fix(-6.8)   ' -6
```

Vous pouvez remarquer que ni `Int()` ni `Fix()` n'arrondit à la valeur supérieure. `Int()` renvoie l'entier inférieur le plus proche de l'argument. Les nombres négatifs sont arrondis à l'entier négatif inférieur le plus proche. `Fix()` renvoie la portion entière tronquée de la valeur, c'est-à-dire que la décimale de l'argument est jetée aux oubliettes.

Tronquer signifie retirer. `Fix()` tronque la portion décimale de son argument. Ainsi, une valeur de 5.23 tronquée donnerait 5 ; une valeur de –5.23 donnerait –5.

La fonction de *valeur absolue* est utile pour calculer la différence de valeurs telles que distances et températures.

La valeur absolue est la valeur positive d'un nombre. Ainsi, la valeur absolue de 19 est 19, et la valeur absolue de –19 est également 19.

`Abs()` est la fonction interne qui permet de calculer les valeurs absolues. Imaginons que vous cherchiez à mesurer la différence d'âge entre deux personnes. La valeur absolue de cette différence d'âge serait ainsi obtenue :

```
intAgeDiff = Abs(intAge1 - intAge2)
```

Quelle que soit la personne la plus âgée, cette instruction ne peut affecter à `intAgeDiff` qu'une valeur positive. Si, d'aventure, `intAge1` était inférieur à intAge2, et que l'on n'ait pas utilisé la fonction `Abs()`, le calcul aurait donné une valeur négative.

La fonction `Sqr()` renvoie la racine carrée d'un nombre positif. Dans les instructions suivantes, les commentaires indiquent le résultat de chaque fonction `Sqr()` :

```
intVal1 = Sqr(4)      ' 2
intVal2 = Sqr(64)     ' 8
intVal3 = Sqr(4096)   ' 16
```

`Sqr()` renvoie également la racine carrée exacte des valeurs décimales.

Visual Basic supporte plusieurs fonctions scientifiques et trigonométriques avancées. Voici une liste partielle :

- `Exp()` renvoie la base d'un logarithme naturel (nombre *e*, valant approximativement 2,718282), élevée à la puissance de l'argument.
- `Log()` renvoie le logarithme naturel de l'argument.
- `Atn()` renvoie la tangente d'arc de l'argument, en radians.
- `Cos()` renvoie le cosinus de l'argument, en radians.
- `Sin()` renvoie le sinus de l'argument, en radians.
- `Tan()` renvoie la tangente de l'argument, en radians.

Pour appliquer une fonction trigonométrique à une valeur en degrés, et non en radians, il faut multiplier l'argument par (π / `180`). Le nombre π (pi) vaut approximativement 3,14159.

Fonctions de type de données

Il existe plusieurs fonctions qui s'occupent du type de données des arguments plutôt que de leur valeur :

- les fonctions d'inspection de données `IsDate()`, `IsNull()`,`IsNumeric()` et `VarType()` ;
- les fonctions abrégées `IIf()` et `Choose()` ;
- les fonctions de conversion de type de données.

Fonctions d'inspection de données

Les fonctions d'inspection de données vérifient le type des données et le contenu des variables. Un programme doit traiter des données de types très variés, et l'on ne sait pas toujours à l'avance sur quoi on va tomber. Avant de procéder à un calcul, par exemple, il convient de s'assurer que les données sont numériques.

Le Tableau 8.1 liste les fonctions `Is()` et détaille leur activité. Chacune de ces fonctions reçoit un argument de type `Variant`.

Tableau 8.1 : Les fonctions Is() vérifient le contenu des variables et des contrôles

Fonction	*Description*
`IsDate()`	Vérifie si son argument est de type `Date` (ou s'il peut être correctement converti en date).
`IsEmpty()`	Vérifie si son argument a été initialisé.
`IsNull()`	Vérifie si son argument contient une valeur `Null`.
`IsNumeric()`	Vérifie si son argument est d'un type numérique (ou s'il peut être correctement converti en nombre).

Si les fonctions `Is...()` acceptent le type de données `Variant`, c'est qu'elles doivent être en mesure d'inspecter et d'identifier des données de n'importe quel type.

La section de code présentée dans le Listing 8.3 est assez simple, mais elle décrit le comportement de la fonction `IsEmpty()` selon que les variables ont ou n'ont pas été initialisées. `IsEmpty()` permet, par exemple, de vérifier si l'utilisateur a entré une valeur dans un champ.

Listing 8.3 : Vérification de variables vides

```
1:  ' Interroge les fonctions Is().
2:  Dim var1 As Variant, var2 As Variant,
3:  Dim var3 As Variant, var4 As Variant
4:  Dim intMsg As Integer   ' valeur de renvoi de MsgBox
5:  ' Affectations de valeurs d'exemple.
6:  var1 = 0        ' Valeur zéro.
7:  var2 = Null     ' Valeur Null.
8:  var3 = ""       ' Chaîne Null.
9:  ' Appelle chaque fonction Is().
10: If IsEmpty(var1) Then
11:    intMsg = MsgBox("var1 est vide.", vbOKOnly)
12: End If
13: If IsEmpty(var2) Then
14:    intMsg = MsgBox("var2 est vide.", vbOKOnly)
15: End If
16: If IsEmpty(var3) Then
17:    intMsg = MsgBox("var3 est vide.", vbOKOnly)
18: End If
19: If IsEmpty(var4) Then
20:    intMsg = MsgBox("var4 est vide.", vbOKOnly)
21: End If
```

La seule sortie de ce code est une boîte de message qui affiche ceci :

```
var4 est vide
```

Pourquoi ? Parce que toutes les autres variables ont un certain type de données — toutes ont été *initialisées*.

C'est `IsNull()` qu'il faut utiliser pour vérifier la présence de données dans des champs ou des contrôles quelconques. `IsEmpty()` ne devrait être appliquée qu'aux variables.

`IsNull()` interroge son argument et renvoie `True` si l'argument contient une valeur `Null`. La valeur `Null` est une valeur spéciale que l'on affecte aux variables pour indiquer l'absence de données, ou pour signaler une erreur. (L'interprétation qu'aura le programme de `Null` dépend, en fait, de votre code.)

Si l'on peut affecter à une variable une valeur `Null` (comme dans `varA = Null`), on ne saurait en revanche interroger la valeur `Null` de cette façon :

```
If (varA = Null) Then...
```

Une telle condition n'est pas vérifiable. `IsNull()` est le seul moyen d'interroger la valeur `Null` d'une variable.

L'interrogation des données est une chose simple. Au cas où votre procédure doit vérifier si une zone de texte `txtHoursWorked` contient ou non des données, une instruction `If` comme celle-ci suffit :

```
If IsNull(txtHoursWorked) Then
   intMsg = MsgBox("Vous n'avez pas entré les heures !", vbOKOnly)
Else
   intMsg = MsgBox("Merci d'avoir entré les heures !", vbOKOnly)
End If
```

L'instruction `If` permet ici de vérifier que l'utilisateur a bien saisi quelque chose dans le champ, avant que le programme ne poursuive son exécution.

`IsNumeric()` cherche dans son argument un nombre. Pour toutes les valeurs `Variant` convertibles en nombres, `IsNumeric()` renvoie `True` ; elle renvoie `False` pour toutes les autres. Voici les types de données convertibles en nombres :

- `Empty` (conversion en `0`) ;
- `Integer` ;
- `Long` ;

- `Single` ;
- `Double` ;
- `Currency` ;
- `Date` (renvoie toujours `False`) ;
- `String`, si la chaîne "semble" un nombre valide.

Le code suivant demande à l'utilisateur son âge, à l'aide d'une variable `Variant`. Si l'utilisateur entre une valeur non numérique, le programme affiche un message d'erreur.

```
1:  Dim varAge As Variant
2:  Dim intMsg As Integer   ' Valeur de renvoi de MsgBox()
3:  varAge = InputBox("Quel âge avez-vous ?", "Age")
4:  If IsNumeric(varAge) Then
5:     intMsg = MsgBox("Merci.", vbOKOnly)
6:  Else
7:     intMsg = MsgBox("Pas de cachotteries !",
    ➥vbOKOnly+vbExclamation)
8:  End If
```

La ligne 4 s'assure que l'utilisateur a bien entré une valeur numérique, plutôt que d'avoir saisi, par exemple, son nom en toutes lettres.

La fonction `VarType()` permet de déterminer le type de données d'une variable. Le Tableau 8.2 présente les seules valeurs que puisse renvoyer une fonction `VarType()`.

Tableau 8.2 : Les valeurs renvoyées par VarType() indiquent le type de données

Valeur renvoyée	*Littéral nommé*	*Type de données*
0	vbEmpty	Empty
1	vbNull	Null
2	vbInteger	Integer
3	vbLong	Long
4	vbSingle	Single
5	vbDouble	Double
6	vbCurrency	Currency
7	vbDate	Date
8	vbString	String
9	vbObject	Object

Tableau 8.2 : Les valeurs renvoyées par VarType() indiquent le type de données *(suite)*

Valeur renvoyée	*Littéral nommé*	*Type de données*
10	vbError	Valeur d'erreur
11	vbBoolean	Boolean
12	vbVariant	Variant*
13	vbDataObject	Objet d'accès aux données
14	vbDecimal	Decimal
17	vbByte	Byte
8192	vbArray	Tableau**

* Sur les tableaux de `Variant`, voyez au Chapitre 10.
** Pour indiquer un tableau, Visual Basic ajoute 8192 à la valeur de type de données ; ainsi, 8194 indique un tableau d'entiers, etc.

Dans le Listing 8.4, la procédure se sert d'une instruction `Select Case` pour afficher le type des données qui lui sont passées.

Listing 8.4 : VarType() permet de déterminer le type des données passées

```
1:  Private Sub PrntType(varA)  ' Variant par défaut.
2:    Dim intMsg As Integer     ' Valeur de renvoi de MsgBox().
3:    Select Case VarType(varA) ' VarType() renvoie un entier.
4:      Case 0
5:        intMsg = MsgBox("L'argument est de type Empty.")
6:      Case 1
7:        intMsg = MsgBox("L'argument est de type Null.")
8:      Case 2
9:        intMsg = MsgBox("L'argument est de type Integer.")
10:     Case 3
11:       intMsg = MsgBox("L'argument est de type Long.")
12:     Case 4
13:       intMsg = MsgBox("L'argument est de type Single.")
14:     Case 5
15:       intMsg = MsgBox("L'argument est de type Double.")
16:     Case 6
17:       intMsg = MsgBox("L'argument est de type Currency.")
18:     Case 7
19:       intMsg = MsgBox("L'argument est de type Date.")
20:     Case 8
21:       intMsg = MsgBox("L'argument est de type String.")
22:     Case 9
23:       intMsg = MsgBox("L'argument est de type Object.")
```

Listing 8.4 : VarType() permet de déterminer le type des données passées *(suite)*

```
24:      Case 10
25:        intMsg = MsgBox("L'argument est de type Error.")
26:      Case 11
27:        intMsg = MsgBox("L'argument est de type Boolean.")
28:      Case 12
29:        intMsg = MsgBox("L'argument est de type tableau de Variant.")
30:     Case 13
31:        intMsg = MsgBox("L'argument est de type objet d'accès
           ➥aux données.")
32:      Case 14
33:        intMsg = MsgBox("L'argument est de type Decimal.")
34:      Case 17
35:        intMsg = MsgBox("L'argument est de type Byte.")
36:      Case Else
37:        intMsg = MsgBox("L'argument est de type Array (tableau).")
38:    End Select
39:  End Sub
```

Les fonctions abrégées *IIf()* et *Choose()*

Il existe un équivalent plus simple de l'instruction `If... Else`. Il s'agit de la fonction `Iif()`, capable de prendre avantageusement la place d'une instruction `If... Else` monoligne. `IIf()` est semblable à la fonction `@If()` que l'on trouve dans les tableurs. Voici son format :

```
IIf(condition, TrueBody, FalseBody)
```

`IIf()` ne doit être utilisé qu'en remplacement d'instructions `If... Else` courtes, telles que celles-ci :

```
If (curSales < 5000.00) Then
   curBonus = 0.00
Else
   curBonus = 75.00
End If
```

Chacun des deux corps de cette instruction `If... Else` étant constitué d'une seule ligne, on peut écrire une fonction `IIf()` plus courte. La suivante affecte la valeur de renvoi à `curBonus` :

```
curBonus = IIf(curSales < 5000.00, 0.00, 75.00)
```

La Figure 8.4 illustre le fonctionnement de cette `IIf()`.

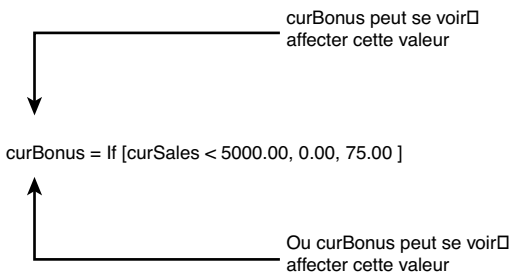


Figure 8.4
L'une des deux valeurs est affectée à la variable située à gauche IIf().

Si l'équivalent `IIf()` est plus court que l'instruction `If... Else` multiligne, il est aussi moins clair. Si, du reste, vous vouliez ajouter quoi que ce soit d'un côté ou de l'autre de la fonction `IIf()`, il faudrait d'abord la convertir en `If...Else` multiligne. Aussi le format `If... Else` multiligne est-il préférable dans la plupart des cas.

On ne peut diviser par zéro (la division par zéro n'est pas définie en mathématique). Si une division par zéro advient, la fonction `IIf()` suivante renvoie un prix de vente moyen ou une valeur `Null` :

```
curAveSales = IIf(intQty > 0, curTotalSales / intQty, Null)
```

Visual Basic interprète toujours la valeur zéro comme un résultat faux. Vous pourriez donc réécrire l'instruction précédente de la manière suivante :

```
curAveSales = IIf(intQty, curTotalSales / intQty, Null)
```

La fonction `Choose()`, quant à elle, offre une version abrégée de certaines instructions `Select Case`. `Choose()` peut avoir beaucoup d'arguments — plus d'arguments même que toutes les autres fonctions internes. Selon la valeur du premier argument, `Choose()` renvoie un seul des autres arguments. Voici le format de `Choose()`:

```
Choose(intIndexNum, expression[, expression] ...)
```

Après le deuxième argument, (*`expression`*), on peut ajouter autant d'arguments qu'il est nécessaire. `int`*`IndexNum`* doit être une variable ou un champ avec une valeur comprise entre 1 et le nombre d'expressions de la fonction.

Si, par exemple, il faut générer une petite table de codes produits ou de prix, `Choose()` est plus approprié qu'une instruction `If` ou `Select Case`. En revanche, la portée de `Choose()` est plus restreinte que celle de `If`, parce que `Choose()` n'effectue pas de véritable comparaison, mais choisit une valeur entière unique.

`Choose()` renvoie `Null` si `intIndexNum` n'est pas compris entre 1 et le nombre d'expressions incluses.

Le premier argument de `Choose()` peut être une expression. La valeur de cet argument doit être incluse dans le nombre des arguments qui suivent. Si, par exemple, les valeurs possibles d'un index vont de 0 à 4, on ajoute 1 à l'index de sorte que la plage aille de 1 à 5, et que le bon choix soit opéré dans la liste de `Choose()`.

Imaginons un feuille sur laquelle un label indique des codes de prix. Lorsque l'utilisateur entre un nouveau produit, il doit aussi saisir un code de prix entre 1 et 5, à savoir :

1	Plein prix
2	Remise de 5 %
3	Remise de 10 %
4	Commande spéciale
5	Commande par correspondance

La fonction `Choose()` suivante affecte à un champ `Descript` la description associée au code :

```
Descript = Choose(lblProdCode, "Plein prix", "Remise de 5 % ",
➥"Remise de 10 %","Commande spéciale", "Commande par
➥correspondance")
```

Fonctions de conversion de type de données

Le Tableau 8.3 décrit les fonctions de conversion de type de données, qui se distinguent par leur préfixe (`C` pour *conversion*). Chaque fonction convertit son argument d'un type de données à un autre.

Tableau 8.3 : Fonctions de conversion de type de données

Fonction	*Description*
`CBool()`	Convertit l'argument en `Boolean`
`CByte()`	Convertit l'argument en `Byte`
`CCur()`	Convertit l'argument en `Currency`

Tableau 8.3 : Fonctions de conversion de type de données *(suite)*

Fonction	*Description*
`CDate()`	Convertit l'argument en `Date`
`CDbl()`	Convertit l'argument en `Double`
`CDec()`	Convertit l'argument en `Decimal`
`CInt()`	Convertit l'argument en `Integer`
`CLng()`	Convertit l'argument en `Long`
`CSng()`	Convertit l'argument en `Single`
`CStr()`	Convertit l'argument en `String`
`CVar()`	Convertit l'argument en `Variant`

Toutes ces fonctions exigent naturellement que l'argument soit convertible. Par exemple, `CByte()` ne pourra convertir le nombre 123456789 en `Byte`, parce que ce type de données est incapable de contenir une aussi grande valeur.

A la différence de `Int()` et `Fix()`, `CInt()` arrondit l'argument à l'entier le plus proche. Pour les valeurs négatives, `CInt()` renvoie également l'entier le plus proche. Dans les instructions suivantes, les commentaires indiquent le contenu de chaque variable :

```
intA1 = CInt(8.5)       ' Affecte la valeur 8 à intA1.
intA2 = CInt(8.5001)    ' Affecte la valeur 9 à intA2.
```

Le code suivant déclare des variables de quatre types de données différents, puis convertit chaque argument à ces types de données. Rappelez-vous que l'on passe aussi bien des expressions numériques à ces fonctions, ce qui permet de définir un type de données avant de stocker le résultat calculé dans une variable ou dans un champ.

```
curVar1 = CCur(123)      ' Convertit 123 en Currency.
dblVar2 = CDbl(123)      ' Convertit 123 en Double.
sngVar3 = CSng(123)      ' Convertit 123 en Single.
varVar4 = CVar(123)      ' Convertit 123 en Variant.
```

Fonctions de chaînes

Les fonctions de chaînes traitent et analysent le contenu des chaînes. Visual Basic a hérité du BASIC l'un de ses plus gros avantages sur les autres langages de programmation : un support efficace des données de chaînes.

La fonction *Len()*

`Len()` est l'une des rares fonctions qui acceptent pour arguments aussi bien les variables numériques que les chaînes. Toutefois, `Len()` est principalement appliquée aux chaînes. Cette fonction renvoie le nombre d'octets qu'occupe en mémoire l'argument. Voici le format de `Len()` :

```
Len(Expression)
```

`Len()` accepte toutes les valeurs de chaînes : variables, littéraux, expressions. Cependant, seules les variables numériques peuvent lui être passées comme arguments — ni littéraux numériques, ni expressions numériques.

`Len()` renvoie la longueur en nombre de caractères de la variable chaîne, de la constante chaîne ou de l'expression chaîne située entre les parenthèses. La fonction `MsgBox()` affiche le résultat `6` :

```
intMsg = MsgBox(Len("abcdef"))
```

Si la chaîne contient `Null`, `Len()` renvoie la valeur `0`. L'interrogation d'une chaîne nulle permet de vérifier si l'utilisateur a entré ou non des données en réponse à une fonction `InputBox()` ou à un contrôle.

Conversion de chaînes

Plusieurs fonctions de conversion s'appliquent aux données de chaînes. Le Tableau 8.4 décrit chacune des fonctions utilisées dans les exemples qui suivent.

Tableau 8.4 : Fonctions de conversion de chaînes

Fonction	*Description*
`CStr()`	Convertit l'argument en chaîne.
`Str()`	Convertit un argument numérique en chaîne (plus précisément en `Variant` exploitable comme chaîne).
`Val()`	Convertit un argument chaîne en nombre (si la chaîne passée contient bien un nombre).

`CStr()` et `Str()` convertissent leurs arguments en chaînes. La différence est que `Str()` fait précéder d'un blanc (espace) les nombres positifs convertis en chaîne. Le Listing 8.5 illustre cette différence entre `CStr()` et `Str()`.

Listing 8.5 : Contrairement à CStr(), Str() fait précéder les nombres positifs d'un blanc

```
1:  Private Sub convStr ()
2:    Dim str1 As String, s2 As String
3:    Dim intMsg As Integer   ' Clic sur le bouton.
4:    str1 = CStr(12345)
5:    str2 = Str(12345)
6:    intMsg = MsgBox("***" & str1 & "***")
7:    intMsg = MsgBox("***" & str2 & "***")
8:  End Sub
```

La ligne 6 génère une boîte de message qui affiche `***12345***`, tandis que la ligne 7 entraîne l'affichage de `*** 12345***`. `Str()` ajoute un blanc avant le nombre.

Fonctions ASCII

`Chr()` et `Asc()` permettent de convertir une chaîne en sa valeur ASCII, et vice versa. La table ASCII recense tous les caractères possibles sur un PC et attribue à chacun un numéro séquentiel (*code ASCII*).

`Chr()` renvoie le caractère correspondant au code ASCII qui lui est passé comme argument. `Chr()` permet ainsi d'afficher des caractères non disponibles sur le clavier, mais figurant dans la table ASCII.

La fonction `Asc()` est l'exact inverse de `Chr()`. Là où `Chr()` reçoit un argument numérique et renvoie un caractère, `Asc()` reçoit un argument chaîne et le convertit en valeurs ASCII correspondantes.

Au terme de l'instruction suivante, `strVar` contient la lettre "A", puisque 65 est le code ASCII de ce caractère.

```
strVar = Chr(65)    ' Stocke "A" dans aVar
```

Bien entendu, une telle instruction n'a en soi aucun intérêt ; il est bien plus simple d'affecter directement à `strVar` le contenu `A`. Et qu'en est-il si, par exemple, votre programme doit poser une question en espagnol ? En espagnol, les questions sont précédées d'un point d'interrogation à l'envers, caractère qui n'apparaît pas sur votre clavier. Avec `Chr()`, vous affichez ce caractère dans votre boîte de message :

```
' Chr(241) donne un "ñ" (n tilde).
strMyQuest = Chr(191) & " Como esta, " & "Se" & Chr(241) & "or ?"
intMsg = MsgBox(strMyQuest)
```

La Figure 8.5 montre la boîte de message générée par ce code.

Figure 8.5
Les codes ASCII permettent d'afficher les caractères non disponibles sur le clavier.

`Asc()` renvoie le code ASCII du caractère qui lui est passé comme argument. (La table ASCII est notamment fournie dans l'aide en ligne de Visual Basic.) L'argument doit être une chaîne d'au moins un caractère. Mais en présence de plusieurs caractères dans la chaîne passée, `Asc()` ne renvoie que le code ASCII du premier caractère.

Voici un bon exemple d'utilisation de `Asc()` :

```
strAns = InputBox("Voulez-vous connaître le nom ?")
If ((Asc(strAns) = 79) Or (Asc(strAns) = 111)) Then
b = MsgBox("Le nom est " + aName)
End If
```

L'utilisateur peut répondre par `o`, `O`, `oui`, `Oui` ou `OUI`. L'interrogation `If...Then` vérifiera avec succès toutes ces entrées, puisque `79` est le code ASCII de `O`, et `111` celui de `o`. Nous disons donc : `Asc()` renvoie la valeur ASCII du premier caractère de la chaîne passée.

Fonctions de sous-chaînes

Les fonctions de sous-chaînes renvoient une partie de la chaîne. `Right()` renvoie les caractères à partir de la droite, `Left()` à partir de la gauche. `Mid()` prélève les caractères à partir du milieu de la chaîne.

Voici le format des fonctions de sous-chaînes :

```
Left(stringValue, numericValue)
Right(stringValue, numericValue)
Mid(stringValue, startPosition[, length])
```

Le code suivant illustre le comportement de `Left()` :

```
strA = "abcdefg"
partSt1 = Left(strA, 1)   ' Renvoie "a".
partSt2 = Left(strA, 3)   ' Renvoie "abc".
partSt3 = Left(strA, 20)  ' Renvoie "abcdefg".
```

Si vous demandez à `Left()` de renvoyer plus de caractères qu'il n'en existe, la chaîne entière est renvoyée.

`Right()` fait la même chose en sens inverse :

```
strA = "abcdefg"
partSt1 = Right(strA, 1)   ' Renvoie "g".
partSt2 = Right(strA, 3)   ' Renvoie "efg".
partSt3 = Right(strA, 20)  ' Renvoie "abcdefg".
```

`Mid()` complète `Left()` et `Right()` en prélevant les caractères à partir du milieu. `Mid()` requiert trois arguments : une chaîne suivie de deux entiers. Le premier entier indique où `Mid()` doit commencer son prélèvement (position à partir de `1`), et le second entier détermine combien de caractères après cette position seront renvoyés. Si vous ne spécifiez pas les deux entiers, `Mid()` prend `1` comme première position.

`Mid()` peut prélever n'importe quel nombre de caractères depuis n'importe quel point de la chaîne. Voici un exemple :

```
strA = "Visual Basic FORTRAN COBOL C Pascal"
lang1 = Mid(strA, 1, 12)  ' Renvoie "Visual Basic".
lang2 = Mid(strA, 14, 7)  ' Renvoie "FORTRAN".
lang3 = Mid(strA, 22, 5)  ' Renvoie "COBOL".
lang4 = Mid(strA, 28, 1)  ' Renvoie "C".
lang5 = Mid(strA, 30, 6)  ' Renvoie "Pascal".
```

Si l'argument *`length`* (longueur) n'est pas spécifié, VB renvoie tous les caractères situés à droite de la position. Si la longueur spécifiée est plus grande que le reste de la chaîne, Visual Basic ignore l'argument *`length`*.

`Mid()` est utilisable à la fois comme fonction et comme commande. Elle est commande lorsqu'on l'emploie à gauche de l'opérateur = d'une instruction d'affectation. Elle est fonction partout ailleurs. Voici son format :

```
Mid(string, start[, length])
```

L'instruction `Mid()` modifie le contenu de la chaîne donnée entre parenthèses. Le code suivant initialise une chaîne avec trois mots, puis change le mot du milieu à l'aide de `Mid()` :

```
strSentence = "Paul et Marie"
' Change le mot du milieu
Mid(strSentence, 6, 2) = "ou"
' Après changement
intMsg = MsgBox("Résultat après changement : " & strSentence)
➥' Affiche "Paul ou Marie"
Inversion de style
```

`UCase()` renvoie l'argument chaîne en lettres capitales. `LCase()` le renvoie en lettres minuscules. La fonction `MsgBox()` suivante affiche `VISUAL BASIC` :

```
intMsg = MsgBox(UCase("Visual Basic"))
```

Les fonctions *LTrim()* et *RTrim()*

`LTrim()` et `RTrim()` suppriment les espaces au début et à la fin de la chaîne. `LTrim()` renvoie l'argument chaîne sans les espaces de début, `RTrim()` sans les espaces de fin. La fonction `Trim()` supprime les espaces de début et de fin.

Voici le format de ces fonctions :

```
LTrim(stringExpression)
RTrim(stringExpression)
Trim(stringExpression)
```

Les instructions suivantes suppriment les espaces de début, de fin, et des deux :

```
st1 = LTrim("     Bonjour")  ' Renvoie "Bonjour".
st2 = RTrim("Bonjour     ")  ' Renvoie "Bonjour".
st3 = Trim("   Bonjour   ")  ' Renvoie "Bonjour".
```

Sans l'intervention des fonctions, le mot "Bonjour" aurait été stocké dans la variable en compagnie de ses diverses espaces.

Lorsque `Str()` convertit un nombre positif en chaîne, une espace est toujours ajoutée au début (pour figurer un signe plus imaginaire). En combinant `LTrim()` à `Str()`, vous éliminez cette espace superflue. Dans la première de ces instructions, la valeur est stockée dans `str1` avec le blanc. Dans la seconde, `LTrim()` supprime le blanc avant d'affecter la valeur à `str2`.

```
str1 = Str(234)           ' Renvoie " 234".
str2 = LTrim(Str(234))    ' Renvoie "234".
```

La fonction `ReverseIt()` suivante inclut plusieurs des fonctions de chaînes présentées ci-dessus. Cette fonction, illustrée dans le Listing 8.6, inverse un certain nombre de caractères dans la chaîne passée.

Listing 8.6 : Cette fonction se sert des fonctions de chaînes pour inverser une chaîne

```
1:  Public Function ReverseIt (strS As String, ByVal n
    ➥As Integer) As String
2:  ' Attend une chaîne, ainsi qu'un entier indiquant
3:  ' le nombre de caractères à inverser.
4:  ' Inverse le nombre spécifié de
```

```
5:  ' caractères dans la chaîne spécifiée.
6:  ' Renvoie la chaîne inversée.
7:  '
8:  ' Inverse les n premiers caractères de la chaîne.
9:
10:     Dim strTemp As String, intI As Integer
11:
12:      If n > Len(strS) Then n = Len(strS)
13:      For intI = n To 1 Step -1
14:          strTemp = strTemp + Mid(strS, intI, 1)
15:      Next intI
16:      ReverseIt = strTemp + Right(strS, Len(strS) - n)
17:  End Function
```

Supposons que la fonction `ReverseIt()` soit appelée par l'instruction suivante :

```
newStr = ReverseIt ("Visual Basic", 6)
```

Si tout se passe bien, la chaîne `newStr` prendra comme contenu `lausiV Basic` (les six premiers caractères sont inversés). La ligne 10 déclare deux variables locales : la première, `strTemp` recevra le contenu inversé de la chaîne ; la seconde, `intI`, est utilisée dans la boucle `For`.

La version 6 de Visual Basic propose une nouvelle fonction de chaîne qui renvoie son argument `Single` inversé. Le Listing 8.6, destiné à illustrer la fonction `Mid()`, aurait été bien plus efficace avec `StrReverse()`.

A la ligne 12, l'instruction `If` s'assure que l'entier passé à `ReverseIt()` n'est pas plus grand que la longueur de la chaîne. Il est impossible d'inverser plus de caractères que n'en contient la chaîne. Si l'argument en question est trop grand, l'instruction `If` l'ajuste, *via* la fonction `Len()`, au nombre réel de caractères, et la chaîne est inversée en entier.

A la ligne 13, la boucle `For` décrémente `n` (la position) jusqu'à la valeur 1. La fonction `Mid()` de la ligne 14 concatène à la nouvelle chaîne (le résultat) le caractère situé à la position `n`. Lorsque `n` atteint 1, les caractères inversés sont envoyés à la nouvelle chaîne (ligne 14). Une fois tous les caractères inversés, le code les concatène à la partie droite de la chaîne passée.

Fonctions spéciales

Visual Basic dispose de fonctions spécifiquement destinées au traitement et à l'analyse des valeurs de date et d'heure. Ces fonctions permettent, par exemple, de savoir à quel moment précis un champ a été édité, notamment pour des questions de sécurité. En outre, les rapports générés par vos applications devraient tous indiquer à quelle date précise on les a produits.

Outre ces fonctions de date et d'heure, il existe des fonctions spéciales de formatage pour afficher une chaîne dans un format donné.

Fonctions de date et d'heure

Ce sont vos paramètres Windows qui déterminent les valeurs renvoyées par `Date` et `Time`. Sur une installation française, la fonction `Date` renvoie la date système dans le type `Variant` (`Date`) et sous ce format :

```
dd-mm-yyyy
```

Ici, *dd* est le jour (entre 01 et 31), *mm* est le mois (entre 01 et 12), et *yyyy* est l'année (entre 1980 et 2099). `Date` étant l'une des rares fonctions se passant d'arguments, elle ne requiert guère de parenthèses.

`Time` renvoie l'heure système dans le type `Variant` (`Date`) et sous ce format :

```
hh:mm:ss
```

Ici, *hh* est l'heure (entre 00 et 23) ; *mm*, les minutes (entre 00 et 59) ; et *ss*, les secondes (entre 00 et 59).

`Now` combine les fonctions `Date` et `Time`. `Now` renvoie une valeur de type `Variant` (`Date`), sous le format suivant :

```
dd/mm/yy hh:mm:ss
```

Il est important de retenir que les fonctions `Date`, `Time` et `Now` renvoient des valeurs qui sont, de façon interne, stockée comme des valeurs à précision double (ce qui assure le stockage correct de la date et de l'heure). Le meilleur moyen de formater les valeurs de date et d'heure est la fonction `Format()`, que nous étudierons à la dernière section de ce chapitre.

A exactement 19 heures 45 minutes, l'instruction `currentTime = Time` stocke `19:45:00` dans la variable `currentTime`.

Si l'on est le 27 août 1999, l'instruction `currentDate = Date` stocke `27/08/99` dans la variable `currentDate`.

Le 27 août 1999, à exactement 19 heures 45 minutes, l'instruction `currentDateTime = Now` stocke `27/08/99 19:45:00` dans la variable `currentDateTime`.

Pour entrer une valeur de date ou d'heure, il faut l'entourer de caractères dièse (#) :

```
#21/11/1993#
```

Parce qu'il existe plusieurs formats de dates, Visual Basic doit pouvoir reconnaître une date sous quelque format que vous l'entriez. Du moment que vous l'encadrez de dièses, vous êtes libre d'utiliser les formats suivants :

```
dd-mm-yy ;
dd-mm-yyyy ;
dd/mm/yy ;
dd/mm/yyyy ;
dd NomMois yyyy ;
dd mmm yyyy (où mmm est une abréviation telle que "oct") ;
dd NomMois yy ;
dd-mmm-yy (où mmm est une abréviation telle que "oct") ;
```

Les heures, quant à elles, peuvent être exprimées de ces façons :

```
hh
hh:mm
hh:mm:ss
```

Fonctions chronométriques

La fonction `Timer` renvoie le nombre de secondes écoulées depuis que l'horloge système a sonné minuit. Le format de `Timer` est on ne peut plus simple :

```
Timer
```

`Timer` est l'une des rares fonctions qui n'acceptent aucun argument (c'est la raison de l'absence de parenthèses). `Timer` est l'outil idéal pour chronométrer un événement. Vous pouvez, par exemple, poser une question à l'utilisateur et mesurer le temps qu'il a pris pour répondre. Pour cela, il faut enregistrer la valeur de `Timer` au moment où la question est posée, puis soustraire cette valeur à celle qu'aura `Timer` au moment de la réponse. La différence entre les deux valeurs représente le nombre de secondes que l'utilisateur a pris pour répondre. Le Listing 8.7 illustre cet exemple.

Listing 8.7 : Chronométrage du temps de réponse de l'utilisateur

```
1:  Public Sub CompTime ()
2:  ' Cette procédure mesure le temps de réponse.
3:    Dim intMsg As Integer   ' Valeur de renvoi de MsgBox().
4:    Dim varBefore, varAfter, varTimeDiff As Variant
5:    Dim intMathAns As Integer
6:    varBefore = Timer    ' Valeur au moment de la question.
7:    intMathAns = Inputbox("Combien font 150 + 235 ?")
8:    varAfter = Timer     ' Valeur au moment de la réponse.
9:    ' La différence entre les deux valeurs représente
10:   ' le temps de réponse de l'utilisateur.
11:   varTimeDiff = varAfter - varBefore
12:   intMsg = MsgBox("Vous avez mis " + Str(varTimeDiff)
    ➥& " secondes !")
13: End Sub
```

La ligne 6 stocke la valeur de `Timer` (le nombre de secondes écoulées depuis minuit) immédiatement avant la question. La ligne 7 pose la question, et dès que l'utilisateur saisit une réponse, la ligne 8 stocke la nouvelle valeur de `Timer`. La différence entre les deux valeurs, calculée à la ligne 11, représente le temps de réponse exact de l'utilisateur.

`Timer` ne s'applique qu'aux délais compris dans une même journée. Les fonctions `DateAdd()`, `DateDiff()` et `DatePart()` viennent combler cette lacune. Le Tableau 8.5 décrit ces trois fonctions arithmétiques.

Tableau 8.5 : Fonctions arithmétiques de comparaison de dates

Fonction	*Description*
`DateAdd()`	Renvoie une nouvelle date après que vous avez ajouté une valeur à une date.
`DateDiff()`	Renvoie la différence entre deux dates.
`DatePart()`	Renvoie une partie d'une date donnée.

Ces fonctions arithmétiques peuvent traiter les éléments de date présentées dans le Tableau 8.6. Les valeurs indiquées sont les arguments utilisés par `DateAdd()`, `DateDiff()` et `DatePart()`.

En dépit de son nom, `DateAdd()` s'applique aussi bien aux dates qu'aux heures (c'est d'ailleurs le cas de toutes les fonctions de date). L'argument passé à `DateAdd()` doit être de type `Date`. Voici le format de `DateAdd()` :

```
DateAdd(interval, number, oldDate)
```

Tableau 8.6 : Arguments des fonctions arithmétiques de date

Valeur	*Période*
`yyyy`	Année
`q`	Trimestre
`m`	Mois
`y`	Jour d'une année
`d`	Jour
`w`	Jour ouvrable (`1` pour dimanche, `2` pour lundi, et ainsi de suite pour `Day()`, `Month()`, `Year()` et `DateDiff()`)
`ww`	Semaine
`h`	Heure
`n`	Minute (notez que ce n'est pas `m`)
`s`	Seconde

Ici, *`interval`* doit être l'une des valeurs (sous forme de chaîne) du Tableau 8.6. Cette argument définit l'unité de temps qui sera ajoutée ou soustraite (secondes, minutes, etc.). L'argument *`number`* spécifie combien de ces unités seront ajoutées ou soustraites. Pour avancer une date, spécifiez un *`interval`* positif ; et un *`interval`* négatif pour retarder la date. *`oldDate`* est la date ou l'heure de départ (la date ou l'heure à laquelle on ajoute ou soustrait). La valeur de *`oldDate`* ne change pas. A la fin, `DateAdd()` renvoie la nouvelle date.

Imaginons que vous réalisez un achat avec une carte de crédit dont le délai de facturation est de vingt-cinq jours. L'instruction suivante ajoute vingt-cinq jours à la date d'aujourd'hui et stocke le résultat dans `intStarts` :

```
intStarts = DateAdd("y", 25, Now)
```

`intStarts` renvoie alors la date dans vingt-cinq jours à partir d'aujourd'hui.

Pour ajouter des jours à une date, on utilise indifféremment `y`, `d` et `w`.

Imaginons maintenant que votre entreprise inscrive les employés à un programme spécial de retraite au bout de dix ans d'ancienneté. L'instruction suivante ajoute dix ans à la date d'embauche et stocke la date résultante dans la variable `anc` :

```
anc = DateAdd("yyyy", 10, emb)
```

La période spécifiée dans la chaîne est ajoutée à la date.

Si vous ne spécifiez pas d'année, toutes les fonctions arithmétiques de date utilisent l'année courante système.

`DateDiff()` renvoie la différence entre deux dates. Pour obtenir une valeur positive, il faut imbriquer `DateDiff()` dans une fonction `Abs()`. Cette différence est exprimée dans l'unité de temps spécifiée. Voici le format de `DateDiff()` :

```
DateDiff(interval, date1, date2)
```

L'instruction suivante détermine le nombre d'années d'ancienneté d'un employé :

```
anc = Abs(DateDiff("yyyy", dateEmb, Now))
```

`DatePart()` renvoie une partie de la date (la partie spécifiée par l'unité de temps). `DatePart()` permet d'extraire d'une date le jour, la semaine, le mois, l'heure, etc. Voici le format de `DatePart()` :

```
DatePart(interval, date)
```

L'instruction suivante stocke le nombre de jours écoulés depuis l'embauche de l'employé :

```
DatePart("w", dateEmb)
```

Les fonctions de date et d'heure que nous avons étudiées traitent des *valeurs sérielles*. Ces valeurs sont stockées comme valeurs à précision double afin que les dates et les heures soient correctement stockées et que les opérations renvoient des résultats corrects.

Une valeur sérielle est une représentation interne de la date ou de l'heure, de `VarType 7` (type de données `Date`) ou `Variant`.

Voici le format de la fonction `DateSerial()` :

```
DateSerial(year, month, day)
```

Ici, *year* est un entier (entre 00 et 99, pour 1900 et 1999, ou une année à quatre chiffres) ou une expression. *month* est un entier (entre 1 et 12) ou une expression. *day* est un entier (entre 1 et 31) ou une expression. En passant une expression, vous pouvez spécifier un nombre d'années, de mois ou de jours, à partir de, ou depuis, la valeur. Pour mieux comprendre ce type d'arguments, examinez les appels de fonctions `DateSerial()` suivants. Chacune d'elles renvoie la même valeur :

```
d = DateSerial(1998, 10, 6)
d = DateSerial(1988+10, 12-2, 1+5)
```

Les fonctions `DateSerial()` permettent de maintenir les arguments de date dans certaines limites. Prenons un exemple. L'année 1996 était bissextile, et le mois de février de cette année n'avait donc que vingt-neuf jours. L'appel de fonction `DateSerial()` suivant semble devoir renvoyer un résultat incorrect, puisque le mois de février, même dans une année bissextile, ne peut contenir trente jours :

```
d = DateSerial(1996, 2, 29+1)
```

Mais cette fonction renverra bien le résultat correct, car `DateSerial()` ajuste la date de sorte que `d` corresponde au 1er mars 1996, soit le jour suivant le dernier jour de février. Le Listing 8.8 illustre une utilisation intéressante de la fonction `DateSerial()`.

Listing 8.8 : Ce code calcule le prochain jour ouvrable après la date spécifiée

```
1:  Function DueDate (dteAnyDate) As Variant
2:  ' Attend une valeur date.
3:  ' Calcule le prochain jour ouvrable
4:  ' après la date spécifiée.
5:  ' Renvoie la date de ce jour-là.
6:
7:      Dim varResult As Variant
8:
9:      If Not IsNull(dteAnyDate) Then
10:        varResult = DateSerial(Year(dteAnyDate),
          ➥Month(dteAnyDate) + 1, 1)
11:        If Weekday(varResult) = 1 Then      ' Dimanche
          ➥ajouter un jour.
12:            DueDate = Result + 1
13:        ElseIf Weekday(varResult) = 7 Then  ' Samedi : ajouter deux jours.
14:            DueDate = varResult + 2
15:        Else
16:            DueDate = varResult
17:        End If
18:      Else
19:          varResult = Null
20:     End If
21: End Function
```

Lorsque cette fonction est appelée, elle reçoit une valeur de date stockée comme `Date` ou `Variant`. Comme l'indiquent les commentaires, la fonction renvoie la date du premier jour ouvrable du mois suivant la date argument (entre `2`, lundi, et `6`, vendredi).

La fonction `DateValue()` diffère seulement de `DateSerial()` en ce qu'elle accepte les arguments de type chaîne. En voici le format :

```
DateValue(stringDateExpression)
```

Ici, *`stringDateExpression`* doit être une chaîne reconnaissable par Visual Basic comme date (voir les exemples donnés plus haut pour l'instruction `Date`). Si l'utilisateur doit entrer une date valeur par valeur (le jour, puis le mois, puis l'année), vous pouvez employer `DateValue()` pour convertir ces valeurs au format sériel interne. Si l'utilisateur doit entrer une date d'un seul tenant (qui sera stockée dans une variable chaîne), telle que `19 octobre 1999`, `DateValue()` convertit également cette chaîne en valeur sérielle de date.

Les fonctions `TimeSerial()` et `TimeValue()` s'appliquent de la même façon, mais aux heures. Si l'utilisateur indique l'heure à l'aide de trois valeurs, `TimeSerial()` convertit ces valeurs au format sériel interne (type `Date` ou `Variant`). Voici le format de `TimeSerial()` :

```
TimeSerial(hour, minute, second)
```

Tout comme `DateSerial()`, `TimeSerial()` accepte pour arguments des expressions, qu'elle ajuste en conséquence.

Quand l'heure est entrée sous forme de chaîne, `TimeValue()` convertit cette chaîne en valeur d'heure, selon ce format :

```
TimeValue(stringTimeExpression)
```

Les fonctions `Day()`, `Month()` et `Year()` convertissent chacune l'argument (de type `Variant` ou `Date`) en numéro du jour, du mois et de l'année. Ces trois fonctions sont fort simples :

```
Day(dateArgument)
Month(dateArgument)
Year(dateArgument)
```

Enfin, `Weekday()` renvoie le numéro du jour ouvrable (voir Tableau 8.6) pour l'argument date qui lui est passé.

Les instructions suivantes passent la date du jour (obtenue par `Now`) aux fonctions `Day()`, `Month()` et `Year()` :

```
d = Day(Now)
m = Month(Now)
y = Year(Now)
```

Les numéros du jour, du mois et de l'année de la date courante sont stockés dans les trois variables.

La fonction *Format()*

`Format()` est l'une des fonctions les plus puissantes et les plus complexes. Elle renvoie l'argument sous un format différent de celui qui a été passé. Voici le format de `Format()` :

```
Format(expression, format)
```

`Format()` renvoie une valeur de type `Variant`, que l'on utilise généralement comme chaîne. L'argument *`expression`* est une expression numérique ou une expression chaîne. `Format()` peut modifier toutes sortes de données : nombres, chaînes, dates, heures, etc. Elle peut notamment servir à afficher un montant en incluant la virgule et le symbole "FF".

L'argument *`format`* est une variable chaîne ou une expression qui contient un ou plusieurs des caractères de formatage présentés aux Tableaux 8.7 à 8.9. Ces trois tableaux correspondent aux diverses catégories de données (chaîne, nombre ou date). Ces tableaux sont longs. Mais il vous suffira de quelques exemples pour comprendre le fonctionnement des caractères de formatage.

Tableau 8.7 : Caractères de formatage de chaînes

Symbole	*Description*
`@`	Un caractère est censé apparaître dans la chaîne à la position de `@`. S'il n'y a pas de caractère à cet endroit, un blanc est inséré. Les `@` (s'il y en a plusieurs) s'appliquent de droite à gauche.
`&`	Semblable à `@`, à ceci près qu'aucun blanc n'est inséré si rien n'apparaît à la place du `&`.
`!`	Inverse l'ordre d'application de `@` et `&` (de gauche à droite, donc).
`<`	Convertit tous les caractères en minuscules.
`>`	Convertit tous les caractères en capitales.

Tableau 8.8 : Caractères de formatage de nombres

Symbole	*Description*
`""` (chaîne nulle)	Le nombre s'affiche sans aucun formatage.
`0`	Un chiffre est censé apparaître dans la chaîne à la position de `0`. S'il n'y a aucun chiffre à cet endroit, c'est `0` qui s'affiche. Si le champ de format contient plus de zéros que n'en contient le nombre à formater, des zéros apparaissent au début ou à la fin. Si le nombre contient plus de positions numériques, `0` ajuste au format demandé toutes les décimales, sans toucher à la partie entière. Ce caractère de formatage sert surtout à insérer des zéros de début et de fin.
`#`	Semblable à `0`, à ceci près que rien n'est inséré si le champ de format contient plus de `#` que le nombre à formater ne contient de chiffres.
`.`	Associé à `0` ou `#`, spécifie le nombre de chiffres qui doit apparaître de part et d'autre du point décimal.
`%`	Multiplie le nombre par 100 et insère le signe de pourcentage % dans la chaîne.
`,`	Placées dans les séries de `0` ou de `#`, les virgules permettent — en notation anglo-saxonne — de séparer les milliers. Une double-virgule indique que le nombre doit être divisé par 1000 (pour réduction d'échelle).
`E-, E+, e-, e+`	Si le format contient au moins un `0` ou un `#`, convertit le nombre en notation scientifique.
`:`	Intercale des deux-points entre les heures, les minutes et les secondes.
`/`	Intercale des slashs entre les jours, les mois et les années.
`-`, `+`, `$`, space	S'affichent dans la chaîne tels quels et à la position donnée.
`\`	Le caractère placé après l'antislash apparaît à sa position dans la chaîne.

Tableau 8.9 : Caractères de formatage de dates

Symbole	*Description*
`c`	Affiche la date (comme `ddddd`, si seule la date apparaît), l'heure (comme `ttttt` si seule l'heure apparaît), ou les deux si les deux valeurs sont présentes.
`d`	Affiche le jour, de `1` à `31`.
`dd`	Affiche le jour sur deux chiffres, soit de `01` à `31`.
`ddd`	Affiche le jour sur trois caractères, soit de `Dim` à `Sam`.
`dddd`	Affiche le jour en toutes lettres, soit de `Dimanche` à `Samedi`.
`ddddd`	Affiche la date selon le Style de date courte spécifié dans les Paramètres régionaux de votre Panneau de configuration (généralement `dd/mm/yy`).
`dddddd`	Affiche la date selon le Style de date longue spécifié dans les Paramètres régionaux de votre Panneau de configuration (généralement `dd mmmm yyyy`).
`w, ww`	Voir Tableau 8.6.
`m`	Affiche le mois, de `1` à `12`. Placé après `h` ou `hh`, `m` représente également les minutes.
`mm`	Affiche le mois sur deux chiffres, soit de `01` à `12`. Placé après `h` ou `hh`, `mm` représente également les minutes.
`mmm`	Affiche le mois sur trois caractères, soit de `Jan` à `Déc`.
`mmmm`	Affiche le mois en toutes lettres, soit de `Janvier` à `Décembre`.
`q`	Affiche le trimestre de l'année.
`y`	Affiche le jour de l'année, de `1` à `366`.
`yy`	Affiche l'année sur deux chiffres, soit de `00` à `99` (pour l'an 2000, `yy` affichera donc `00`).
`yyyy`	Affiche l'année avec tous les chiffres, soit de `1000` à `9999`.
`h, n, s`	Voir Tableau 8.6.
`ttttt`	Affiche l'heure selon le Style de l'heure spécifié dans les Paramètres régionaux de votre Panneau de configuration (généralement `hh:nn:ss`).

`AMPM`, `ampm`, `AP` et `ap` affichent de diverses manières l'heure au format anglo-saxon AM/PM.

Les instructions suivantes mettent en œuvre les caractères de formatage de chaînes. Les commentaires indiquent les valeurs formatées affectées aux variables cibles.

```
strS = Format("AbcDef", ">")    ' ABCDEF
strS = Format("AbcDef", "<")    ' abcdef
strS = Format("00143646421", "(@) @@-@@-@@-@@-@@")
➥' (0) 01-43-64-64-21
```

Comme le montre la dernière instruction, les chaînes peuvent recevoir les formats les plus variés. Si la chaîne à formater, par exemple un numéro de téléphone, est une variable chaîne issue d'un champ de formulaire ou d'un tableau de données, `Format()` s'applique de la même manière.

Imaginons le cas où le code d'appel extérieur (par exemple 0) est facultatif. `Format()` s'appliquant de droite à gauche, l'instruction :

```
strS = Format("0143646421", "(@) @@-@@-@@-@@-@@")
```

stockera dans `strS` le numéro :

```
( ) 01-46-64-64-21
```

Si le code d'appel extérieur avait été saisi, il aurait été affiché entre les parenthèses.

`!` a pour effet d'inverser le sens d'application du formatage ; il ne doit être utilisé que dans le cas, par exemple, où des données sont susceptibles de manquer à la fin de la chaîne. L'instruction :

```
strS = Format("43646421", "!(@) @@-@@-@@-@@"
```

stockera dans `strS` un numéro incorrect :

```
(4) 36-46-42-1
```

Le Listing 8.9 illustre le fonctionnement du formatage numérique. Les commentaires indiquent le résultat du formatage.

Listing 8.9 : Format() appliqué aux nombres

```
1:  strS = Format(9146, "|######|")  ' |9146|
2:  strS = Format(2652.2, "00000.00")  ' 02652.20
3:  strS = Format(2652.2, "#####.##")  ' 2652.2
4:  strS = Format(2652.216, "#####.##")  ' 2652.22
5:  strS = Format(45, "+###")  ' +45
6:  strS = Format(45, "-###")  ' -45
```

```
7:  strS = Format(45, "###-")  ' 45-
8:  strS = Format(2445, "####.## FF")  ' 2445. FF
9:  strS = Format(2445, "####.00 FF")  ' 2445.00 FF
10: strS = Format(2445, "00H00")  ' 24H45
```

Le Listing 8.10 illustre le fonctionnement du formatage des dates et des heures. Les commentaires indiquent le résultat du formatage.

Listing 8.10 : Format() appliqué aux dates et heures

```
1:  Dim varD As Variant
2:  varD = Now     ' Suppose comme date fictive
3:                 ' le 21 mai 1999 à 12:30 précises.
4:  strND = Format(varD, "c") ' 21/5/99 12:30:00
5:  strND = Format(varD, "w") ' 6
6:  strND = Format(varD, "ww")' 22
7:  strND = Format(varD, "dddd") ' vendredi
8:  strND = Format(varD, "q") ' 2
9:  strND = Format(varD, "hh") ' 12
10: strND = Format(varD, "d mmmm h:nn:ss")  ' "21 mai 12:30:00"
```

En résumé

Ce chapitre a détaillé la structure générale d'un programme Visual Basic. Lorsque vous créez une application contenant plusieurs modules et procédures, vous devez tenir compte de la portée des variables afin que les procédures aient accès à toutes les données nécessaires. Les variables, nous l'avons vu, doivent rester locales le plus souvent possible. Pour se partager les données, vos procédures devront donc se passer les arguments adéquats. Les procédures que vous écrirez pourront être des sous-routines aussi bien que des fonctions. En créant ces procédures, vous construirez vos propres bibliothèques de routines, exploitables dans d'autres applications.

Pour épauler vos propres procédures, Visual Basic met à votre disposition une multitude de fonctions internes, capables d'analyser et de traiter des nombres, des chaînes et d'autres données. Les fonctions internes étant, comme leur nom l'indique, une partie du langage Visual Basic, elles sont disponibles dans tous les modules, à tout moment.

Le prochain chapitre revient à la nature visuelle de Visual Basic, et vous explique comment créer des boîtes de dialogue standards.

Questions-réponses

Q Pourquoi les contrôles ne peuvent-ils être locaux ou globaux ?

R Les contrôles doivent être accessibles à la totalité du code. Ils sont donc, pour ainsi dire, publics pour toutes les applications. Les contrôles sont, en fait, des objets bien distincts du code. Ainsi, à moins que vous ne créiez des variables de contrôles pour y stocker les valeurs de propriétés de certains contrôles, vous n'avez jamais à vous soucier de la portée des contrôles.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quelles variables ont la plus grande portée : les locales, celles de niveau module ou les publiques ?
2. Quelles variables ont la portée la plus restreinte : les locales, celles de niveau module ou les publiques ?
3. Le mot clé `ByRef` est optionnel. Vrai ou faux ?
4. Combien de valeurs une sous-routine peut-elle renvoyer ?
5. Nommez des fonctions qui peuvent constituer un équivalent abrégé de `If`.
6. Que se passe-t-il si le premier argument de `Choose()` est inférieur à 1 ?
7. A quoi sert la fonction `Abs()` ?
8. Dans chacune des instructions suivantes, que recevra la variable `strS` ?

```
a. strS = Format("74135", "&&&&&-&&&&")
b. strS = Format(12345.67, "######.###")
```

9. Sans consulter la table ASCII, dites ce que `intN` contiendra après exécution de l'instruction d'affectation suivante :

```
intN = Asc(Chr(192))
```

10. Quelle est la différence entre les fonctions `Now` et `Time` ?

Exercices

1. Réécrivez le Listing 8.1 de sorte que `SalesTax()` soit une fonction qui renvoie à la procédure appelante le montant total de la taxe. La procédure appelante, `GetTotal()`, devra afficher dans une boîte de message la valeur renvoyée par `SalesTax()`.
2. Réécrivez en fonction `IIf()` l'instruction `If` suivante :

```
If (intTotal > 10000) Then
   strTitle = "Bon boulot !"
Else
   strTitle = "Viré !"
End If
```

3. Réécrivez en fonction `Choose()` l'instruction `If` suivante :

```
If (ID = 1) Then
  intBonus = 50
ElseIf (ID = 2) Then
  intBonux = 75
ElseIf (ID = 3) Then
  intBonus = 100
End If
```

4. Dans les instructions suivantes, quelles sont les valeurs affectées ?

```
intN = Int(-5.6)
intO = Fix(-5.6)
intP = CInt(-5.6)
```

Chapitre 9

Les boîtes de dialogue

Dans ce chapitre, vous apprendrez à ajouter des boîtes de dialogue à vos applications. Vous découvrirez le contrôle Common Dialog, qui permet de créer six sortes de boîtes de dialogue standards. Le contrôle Common Dialog permet d'afficher des boîte de dialogue familières à l'utilisateur, qui choisira sans peine parmi une liste de fichiers ou imprimera facilement un rapport depuis votre programme.

Voici ce que nous étudierons aujourd'hui :

- l'importance des boîte de dialogue communes ;
- comment disposer le contrôle Common Dialog ;
- les méthodes du contrôle Common Dialog ;
- les propriétés des boîtes de dialogue ;
- comment répondre aux boîtes de dialogue ;
- l'instruction `On Error Goto`, qui permet de gérer avec simplicité le bouton Annuler.

Les boîtes de dialogue communes

Plus votre programme se conforme à l'apparence générale des applications Windows populaires, telles que Microsoft Word, plus rapidement vos utilisateurs s'y adapteront. Si vous comptez vendre vos programmes, il est capital de séduire l'utilisateur, et notamment de le convaincre d'acheter les futures mises à jour. Et si, plutôt que de lancer vos programmes sur le marché, vous développez des applications pour une société précise, l'important reste ; un programme convivial et cohérent signifie pour vous moins de maintenance et plus de productivité.

Pour créer une application qui, entre autres tâches, ouvre des fichiers et envoie des impressions, un choix entre deux partis s'offre à vous :

- imiter les boîtes de dialogue apparentées des autres applications ;
- innover et tenter d'améliorer les styles de boîtes de dialogue standards.

Rien ne vous empêche, par exemple, d'innover dans le type de boîte de dialogue qui s'affiche lorsque l'utilisateur sélectionne Fichier, Ouvrir. Mais ce serait imprudent. Car votre application ne serait déjà plus standard, et vos utilisateurs devraient revenir sur les habitudes acquises dans les autres applications, même pour effectuer une opération très simple. En outre, de tels programmes demanderaient plus de travail, car vous devriez concevoir de nouvelles boîtes de dialogue pour toutes les manipulations.

Faire
Vos applications doivent présenter les mêmes menus et les mêmes boîtes de dialogue que les applications Windows standards.

Pour ceux qui veulent gagner du temps et se conformer aux standards, Visual Basic dispose du contrôle Common Dialog, qui permet de générer des boîtes de dialogue avec le moins de programmation possible. Ces boîtes de dialogue auront exactement le même aspect et le même fonctionnement que les boîtes de dialogue similaires des applications Windows standards. Voici les options Common Dialog :

- **Couleurs.** Affiche une boîte de dialogue dans laquelle l'utilisateur peut choisir sur une palette de couleurs, et même définir des couleurs personnalisées.
- **Police.** Affiche une boîte de dialogue dans laquelle l'utilisateur choisit parmi des polices, des attributs et des tailles de caractères.
- **Ouvrir.** Affiche une boîte de dialogue dans laquelle l'utilisateur sélectionne un fichier à ouvrir à partir des dossiers et des lecteurs, voire d'un réseau.
- **Imprimer.** Affiche une boîte de dialogue dans laquelle l'utilisateur sélectionne une imprimante ou en modifie les paramètres.
- **Enregistrer.** Affiche une boîte de dialogue dans laquelle l'utilisateur spécifie le nom et le lieu d'enregistrement d'un fichier.
- **Aide Windows.** Lance le moteur d'aide Windows et affiche la boîte de dialogue initiale du système d'aide de votre application.

Le contrôle Common Dialog vous permet d'ajouter diverses boîtes de dialogue à votre application, avec un effort minimum.

Vous pouvez, aussi bien, créer des boîtes de dialogue standards sans l'aide du contrôle Common Dialog. Vous devrez alors reproduire sur la feuille la disposition exacte des zones de texte, barres de défilement, zones de liste, et autres éléments inévitables. Même avec très peu de contrôles sur la feuille, l'écriture d'une boîte de dialogue est toujours fastidieuse. Bref, il faut vraiment avoir une excellente raison pour se passer du contrôle Common Dialog !

Les boîtes de dialogue générées par le contrôle Common Dialog sont dites *modales*.

Est modale une boîte de dialogue que l'utilisateur doit nécessairement fermer (en cliquant sur OK ou sur Annuler) avant de poursuivre toute autre opération dans l'application.

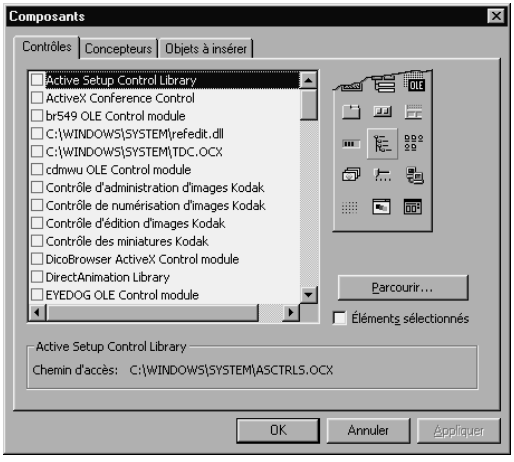
Ajouter le contrôle Common Dialog

Inutile de chercher le contrôle Common Dialog dans votre Boîte à outils, vous ne l'y trouverez pas. La Boîte à outils ne contient pas tous les contrôles disponibles ; d'abord, parce qu'elle occuperait la moitié de l'écran ; ensuite, parce que vous n'avez pas besoin de tous ces contrôles tout le temps. Mais avant d'utiliser le contrôle Common Dialog, il va bien falloir l'ajouter à la Boîte à outils.

Pour ajouter les contrôles, suivez ces étapes :

1. Appuyez sur Ctrl-T (raccourci clavier pour Projet, Composants). La boîte de dialogue Composants s'affiche (voir Figure 9.1).

Figure 9.1
La boîte de dialogue Composants liste tous les contrôles disponibles sur votre système.



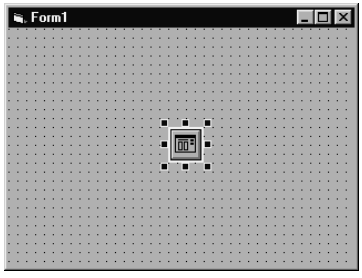
2. Faites défiler la liste jusqu'à l'option *Microsoft Common Dialog Control 6.0*.
3. Cochez la case et cliquez sur OK. Le contrôle Common Dialog s'affiche au bas de la fenêtre Boîte à outils.

Profitez-en pour jeter un coup d'œil aux autres contrôles disponibles dans la boîte de dialogue Composants. Tous sont des contrôles ActiveX. Microsoft Calendar Control 8.0, par exemple, permet d'ajouter à vos applications des fonctions de calendrier, telles qu'en exigerait un programme de gestion de planning ou un programme de facturation. Si vous décidez d'ajouter de nouveaux contrôles à votre Boîte à outils (ça ne coûte rien), vous pouvez également parcourir leurs propriétés spécifiques. Consultez l'aide en ligne au sujet des événements et des méthodes supportés par ces contrôles. Vous trouverez dans les revues spécialisées et sur l'Internet des contrôles supplémentaires capables d'accélérer le développement de vos programmes. Le Chapitre 17 vous apprendra à écrire vos propres contrôles ActiveX.

Fonctionnement du contrôle Common Dialog

La Figure 9.2 montre un contrôle Common Dialog placé au milieu d'une feuille. Comme vous pouvez le constater, il ne ressemble pas vraiment à une boîte de dialogue. Il semble même trop petit pour servir à quoi que ce soit. Et Visual Basic ignore vos tentatives de redimensionnement du contrôle, même s'il est entouré de poignées.

Figure 9.2
Placé sur la feuille, le contrôle Common Dialog ne ressemble pas à grand-chose.



A l'exécution du programme, le contrôle Common Dialog prend la forme de l'une des boîtes de dialogue évoquées plus haut. En fait, Visual Basic se charge d'afficher la boîte de dialogue au centre de l'écran, quelle que soit la position du contrôle Common Dialog

sur la feuille. Vous pouvez donc le placer à l'écart, et disposer les autres contrôles comme bon vous semble. Lorsque le contrôle Common Dialog sera déclenché, la boîte de dialogue s'affichera automatiquement au centre de l'écran. Le reste du temps, Visual Basic masque le contrôle Common Dialog de sorte qu'il ne se superpose pas, lors de l'exécution, aux autres contrôles de la feuille.

Pour que le programme déclenche le contrôle Common Dialog en affichant un type spécifique de boîte de dialogue, vous devez écrire votre code de façon appropriée.

C'est à l'aide des propriétés et des méthodes que l'on spécifie quelle forme de boîte de dialogue le contrôle Common Dialog doit afficher. Les méthodes, vous vous en souvenez peut-être, sont des procédures internes appliquées à des contrôles particuliers. Voici les méthodes du contrôle Common Dialog :

- `ShowColor`. Affiche une boîte de dialogue Couleur.
- `ShowFont`. Affiche une boîte de dialogue Police.
- `ShowHelp`. Affiche une boîte de dialogue Aide Windows.
- `ShowOpen`. Affiche une boîte de dialogue Ouvrir.
- `ShowPrinter`. Affiche une boîte de dialogue Imprimer.
- `ShowSave`. Affiche une boîte de dialogue Enregistrer.

Soit un contrôle Common Dialog nommé `dbFont`. Après quelques réglages de propriétés, il suffit d'appliquer à ce contrôle la méthode `ShowFont` pour afficher une boîte de dialogue de type Police :

```
dbFont.ShowFont   ' Affiche une boîte de dialogue Police
```

De même, pour afficher — par exemple, en réponse à une sélection de menu Fichier, Ouvrir — une boîte de dialogue de type Ouvrir, il suffit d'appliquer la méthode `ShowOpen` au contrôle `dbFont` :

```
dbFont.ShowOpen   ' Affiche une boîte de dialogue Ouvrir
```

Comme vous le voyez, un seul contrôle Common Dialog suffit pour afficher divers types de boîtes de dialogue. Toutefois, avant de déclencher le contrôle par une méthode, il est indispensable de définir quelques propriétés.

Pour les applications réclamant plusieurs types de boîtes de dialogue, deux solutions s'offrent à vous. Ou bien vous placez un seul contrôle Common Dialog et lui appliquez différentes méthodes (comme dans l'exemple ci-dessus). Ou bien vous placez plusieurs contrôles Common Dialog, un pour chaque type de boîte de dialogue requis. Un seul et même contrôle Common Dialog est plus

facile à gérer. Toutefois, le fait de placer plusieurs contrôles Common Dialog vous épargne de modifier les propriétés chaque fois qu'un type de boîte de dialogue est requis.

Les boîtes de dialogue générées par le contrôle Common Dialog ne font rien d'autre que proposer diverses sélections à l'utilisateur. En d'autres mots, si l'utilisateur sélectionne une police de caractères dans la boîte de dialogue Police, puis qu'il clique sur OK, le texte affiché à l'écran ne bougera pas pour autant. Pas plus qu'un fichier ne s'ouvrira lorsque l'utilisateur aura opéré son choix dans la boîte de dialogue Ouvrir. Le contrôle Common Dialog fournit une interface commune pour les boîte de dialogues et définit des propriétés selon les réponses de l'utilisateur. Son rôle s'arrête là. C'est à vous, à votre code, qu'il incombe d'analyser les valeurs de propriétés des boîtes de dialogue, et de déterminer la réaction du programme.

La boîte de dialogue Couleur

Les couleurs disponibles sur Windows vont de plusieurs centaines à plusieurs millions. La boîte de dialogue Couleur offre à l'utilisateur un moyen simple de choisir parmi ces teintes. Supposons, par exemple, que vous permettiez à l'utilisateur de changer la couleur de l'arrière-plan d'une feuille. Il n'est pas question de le demander ainsi :

```
strAns = InputBox("Quelle couleur voulez-vous appliquer à
➥l'arrière-plan ? ")
```

Car la propriété `BackColor`, comme vous vous en souvenez, appelle un code de couleur hexadécimal. (C'est d'ailleurs le cas de toutes les propriétés liées à la couleur, telles que `ForeColor`.) L'utilisateur peut toujours répondre à la `InputBox()` par "rouge" ou "bleu", la pauvre propriété `BackColor` n'y comprendra rien. La réponse de l'utilisateur ne peut en aucun cas être affectée ainsi :

```
frmTitle.BackColor = strAns    ' Cela ne marchera JAMAIS !
```

La boîte de dialogue Couleurs offre à l'utilisateur un moyen de sélectionner une couleur, mais se charge également de convertir la valeur choisie en code hexadécimal. Une fois la boîte de dialogue refermée, il ne vous reste qu'à affecter la valeur renvoyée à la propriété `BackColor` de la feuille.

Pour que le contrôle Common Dialog affiche la boîte de dialogue Couleurs, suivez ces étapes :

1. Dans la propriété `DialogTitle` du contrôle Common Dialog, saisissez le libellé qui doit apparaître sur la barre de titre. Par exemple, `Couleur d'arrière-plan`.

2. Affectez à la propriété `Flags` du contrôle l'une des valeurs présentées au Tableau 9.1.

Tableau 9.1 : Valeurs de la propriété Flags pour la boîte de dialogue Couleurs

Littéral nommé	*Valeur*	*Description*
`cdlCCRGBInit`	`1`	Définit la valeur initiale
`cdlCCFullOpen`	`2`	Affiche la boîte de dialogue complète (incluant la section Définition de couleurs personnalisées)
`cdlCCPreventFullOpen`	`4`	Désactive l'option Définition de couleurs personnalisées
`cdlCCHelpButton`	`8`	Affiche un bouton Aide dans la boîte de dialogue

   Ce tableau décrit l'aspect premier de la boîte de dialogue Couleurs. Pour définir plusieurs de ces valeurs, il suffit de les additionner.

3. Déclenchez l'affichage de la boîte de dialogue Couleurs en appliquant, dans le code, la méthode `ShowColor` au contrôle Common Dialog.

Supposons que vous vouliez afficher la boîte de dialogue Couleurs, laisser l'utilisateur opérer son choix, et afficher un bouton d'aide. Supposons encore que le contrôle Common Dialog soit nommé `cdbColor`. Voici à quoi ressemblerait le code :

```
' Définit les propriétés Flags.
cdbColor.Flags = cdlCCFullOpen + cdlCCHelpButton ' Boîte de dialogue complète.
' Affiche la boîte de dialogue Couleur.
cdbColor.ShowColor
```

La Figure 9.3 montre la boîte de dialogue résultante.

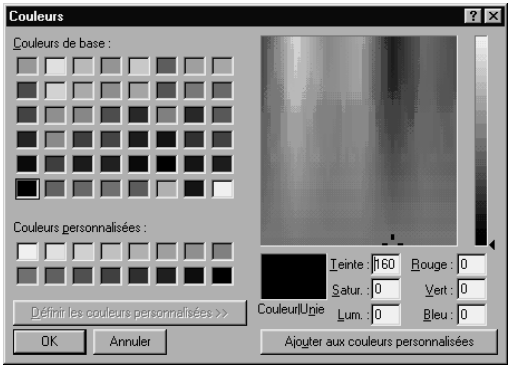
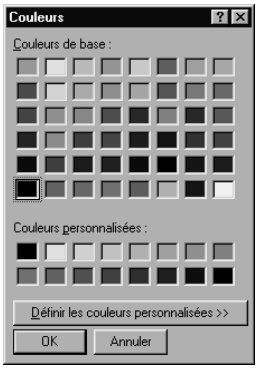


Figure 9.3
La boîte de dialogue Couleur, affichée par la méthode ShowColor.

Pour afficher par défaut un jeu de couleur restreint, tout en laissant à l'utilisateur la possibilité de cliquer sur le bouton Définition de couleurs personnalisées, il suffit d'omettre la valeur `cdlCCFullOpen`. La Figure 9.4 montre cette boîte de dialogue restreinte.

Figure 9.4
La boîte de dialogue Couleur en mode restreint.



Dès que la boîte de dialogue se referme, les propriétés du contrôle sont initialisées en fonction de l'utilisateur. La propriété la plus importante est `Color`, qui contient dès lors la valeur hexadécimale correspondant à la couleur sélectionnée ou créée. Le code suivant pourrait faire suite à l'affichage de la boîte de dialogue :

```
' Définit la couleur d'arrière-plan de la
' feuille selon les choix de l'utilisateur.
frmTitle.ForeColor = cdbColor.Color
```

Gestion du bouton Annuler

Votre code doit être en mesure de déterminer si l'utilisateur a sélectionné une couleur puis cliqué sur OK, ou s'il a cliqué sur le bouton Annuler — ce qui signifie qu'aucune propriété ne doit être modifiée. Cette nécessité de gérer le bouton Annuler ne concerne pas seulement la boîte de dialogue Couleurs, mais aussi toutes les autres boîtes de dialogue.

Pour savoir si l'utilisateur a cliqué sur Annuler, vous emploierez une nouvelle commande Visual Basic : l'instruction `On Error Goto`. Si une erreur quelconque survient dans les instructions qui suivent, cette instruction bascule automatiquement le code vers l'*étiquette* appropriée. Ainsi, l'étiquette :

```
On Error Goto dbErrHandler
```

indique à Visual Basic de passer directement à l'étiquette `dbErrHandler` si une erreur se produit dans les lignes qui suivent (jusqu'à la fin de la procédure).

Une étiquette identifie dans le code une section destinée à gérer les erreurs. On applique aux étiquettes les mêmes conventions de dénomination que pour les variables. En revanche, les étiquettes doivent toujours se terminer par un deux-points, ce qui justement les distingue des variables. Selon l'exemple donné ci-dessus, la procédure doit inclure, quelque part après l'instruction `On Error Goto`*, une étiquette telle que :*

```
dbErrHandler:
```

(Les programmeurs placent en général leurs étiquettes au bas du programme.)

Si une erreur survient, l'instruction `Exit` interrompt la procédure, et le code de gestion d'erreurs qui suit l'étiquette s'exécute. Si l'utilisateur clique sur le bouton Annuler et que vous ayez défini la propriété `CancelError` comme `True`, Visual Basic déclenche une erreur. Bien sûr, un clic sur le bouton Annuler ne constitue pas en soi une erreur. Mais le fait de le traiter comme une condition d'erreur permet d'utiliser un code tel que celui du Listing 9.1.

Listing 9.1 : Gestion du bouton Annuler

```
1:  Private Sub mnuViewColor_Click()
2:    cdbColor.CancelError = True  ' Un clic sur Annuler
3:                                 ' équivaut à une erreur.
4:    On Error Goto dbErrHandler  ' Bascule vers l'étiquette en cas d'erreur.
5:
6:    ' Définit la propriété Flags.
7:    cdbColor.Flags = cdlCCFullOpen + cdlCCHelpButton ' Affichage complet.
8:  Color DB
9:    ' Affiche la boîte de dialogue Couleur.
10:   cdbColor.ShowColor
11:
12:  ' Définit la couleur d'arrière-plan de la
13:  ' feuille selon les choix de l'utilisateur.
14:    frmTitle.ForeColor = cdbColor.Color
15:    Exit Sub   ' Fin de la procédure normale.
16:  dbErrHandler:
17:    ' L'utilisateur ayant cliqué sur Annuler,
18:    ' la procédure doit être ignorée.
19:    Exit Sub
20: End Sub
```

Si l'utilisateur sélectionne une couleur et clique sur OK, la ligne 14 applique cette couleur à l'arrière-plan de la feuille. Plutôt que d'interrompre la procédure quand l'utilisateur

clique sur OK, il est plus simple de définir les couleurs par défaut (entre les lignes 16 et 19) de l'arrière-plan de la feuille.

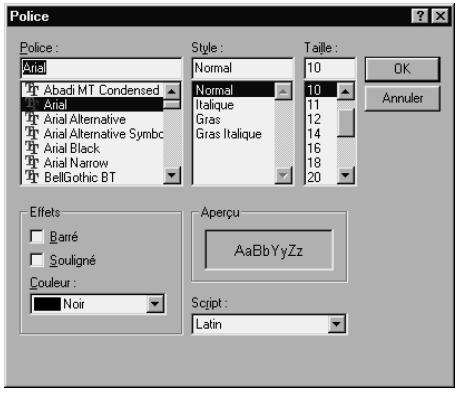
Le gestionnaire d'erreurs de la ligne 16 s'exécutera en réaction à toutes les erreurs, et pas seulement si l'utilisateur clique sur Annuler. Au Chapitre 16, vous découvrirez l'objet système `Err` *à interroger pour déterminer avec précision le problème, sur la base d'un numéro d'erreur.*

La boîte de dialogue Police

Le contrôle Common Dialog permet de générer les boîtes de dialogue Police communes à la plupart des applications. S'il vaut mieux utiliser cette boîte de dialogue plutôt que d'en écrire une vous-même, ce n'est pas seulement parce qu'elle est standard. C'est aussi que vous ne pouvez savoir quelles polices de caractères seront installées sur le PC de l'utilisateur. La boîte de dialogue Police générée par le contrôle Common Dialog recense toutes les polices installées, et affiche le contenu de la boîte de dialogue en conséquence.

La Figure 9.5 montre une boîte de dialogue Police classique, telle que l'affiche la méthode `ShowFont` du contrôle Common Dialog.

Figure 9.5
Une boîte de dialogue Police standard.



Comme pour la boîte de dialogue Couleurs, il faut régler les valeurs de la propriété `Flags`. Toutefois, la boîte de dialogue Police étant plus complexe, les valeurs de `Flags` sont ici différentes.

Pour la boîte de dialogue Police, l'ensemble des valeurs de la propriété `Flags` *peut devenir considérable. On recourt donc soit à des constantes nommées, soit à des valeurs hexadécimales. Le Tableau 9.2 présente les valeurs définissables pour la propriété* `Flags`*. Comme pour la plupart des tâches de programmation, la maintenance est facilitée par l'emploi de littéraux nommés. Le nom des littéraux est généralement explicite, là où les valeurs hexadécimales ne présentent qu'une obscure suite de caractères.*

Tableau 9.2 : Valeurs de Flags pour la boîte de dialogue Police

Littéral nommé	*Valeur*	*Description*
`cdlCFANSIOnly`	`&H400`	La boîte de dialogue n'affichera que des caractères Windows standards
`cdlCFApply`	`&H200`	Affiche le bouton Applique
`cdlCFBoth`	`&H3`	Recense les polices écran ou imprimante disponibles ; la propriété `hDC` identifie le contexte de périphérique associé à l'imprimante
`cdlCFEffects`	`&H100`	Rend disponible les options Barré, Souligné, les attributs de couleur, etc.
`cdlCFFixedPitchOnly`	`&H4000`	L'utilisateur ne pourra sélectionner que des polices à espacement fixe
`cdlCFForceFontExist`	`&H10000`	Affiche un message d'erreur si l'utilisateur tente de sélectionner une police ou un style inexistants
`cdlCFHelpButton`	`&H4`	Affiche le bouton Aide
`cdlCFLimitSize`	`&H2000`	Restreint les tailles de police à la plage définie par les propriétés `Min` et `Max`
`cdlCFNoFaceSel`	`&H80000`	Aucune police sélectionnée par défaut
`cdlCFNoSimulations`	`&H1000`	Interdit les simulations de polices graphiques indépendantes du périphérique (GDI)
`cdlCFNoSizeSel`	`&H200000`	Aucune taille de police sélectionnée par défaut
`cdlCFNoStyleSel`	`&H100000`	Aucun style de police sélectionné par défaut
`cdlCFNoVectorFonts`	`&H800`	Interdit la sélection de polices vectorielles

Tableau 9.2 : Valeurs de Flags pour la boîte de dialogue Police *(suite)*

Littéral nommé	*Valeur*	*Description*
`cdlCFPrinterFonts`	`&H2`	Recense uniquement les polices imprimante (spécifiées par la propriété `hDC`)
`cdlCFScalableOnly`	`&H20000`	N'autorise que la sélection de polices dimensionnables
`cdlCFScreenFonts`	`&H1`	Recense uniquement les polices écran supportées par le système
`cdlCFTTOnly`	`&H40000`	N'autorise que la sélection de polices TrueType
`cdlCFWYSIWYG`	`&H8000`	N'autorise que la sélection de polices imprimante ou écran (cette valeur implique que les valeurs `cdlCFBoth` et `cdlCFScalableOnly` soient également définies)

Les propriétés `cdlCFScreenFonts`, `cdlCFPrinterFonts` et `cdlCFBoth` doivent obligatoirement être définies pour que la boîte de dialogue Police s'affiche. Si vous tentez d'appliquer la méthode `ShowFont`, alors que ces valeurs de `Flags` n'ont pas été définies, Visual Basic génère une erreur.

Le Listing 9.2 met en œuvre la boîte de dialogue Police.

Listing 9.2 : Afficher la boîte de dialogue Police pour que vos utilisateurs choisissent dans une liste le style et la taille de la police

```
1:  ' Définir les valeurs de Flags.
2:  CdbFont.Flags = cdlCFBoth Or cdlCFEffects
3:  CdbFont.ShowFont  ' Affiche la boîte de dialogue.
4:  ' Définit les propriétés du label qui
5:  ' reflétera les choix de l'utilisateur.
6:  LblMessage.Font.Name = CdbFont.FontName
7:  LblMessage.Font.Size = CdbFont.FontSize
8:  LblMessage.Font.Bold = CdbFont.FontBold
9:  LblMessage.Font.Italic = CdbFont.FontItalic
10: LblMessage.Font.Underline = CdbFont.FontUnderline
11: LblMessage.FontStrikethru = CdbFont.FontStrikethru
12: LblMessage.ForeColor = CdbFont.Color
```

Examinez les affectations composites des lignes 6 à 10. Vous n'avez pas encore rencontré cette séparation des noms de propriétés par des points. Rappelez-vous que, lorsque vous cliquez sur les points de suspension de la propriété `Font`, une boîte de dialogue s'affiche et

propose les différentes valeurs. C'est que la propriété `Font` peut contenir une multitude de valeurs. A chaque valeur de la propriété `Font` correspond un style de police, une taille, une couleur, etc. Les noms composites doivent être lus de droite à gauche. Reprenons l'instruction de la ligne 8 :

```
LblMessage.Font.Bold = CdbFont.FontBold
```

Cette instruction ordonne à Visual Basic d'affecter la propriété `FontBold` de la boîte de dialogue (valeur `True` ou `False`) à l'attribut `Bold` de la propriété `Font` du label `lblMessage`.

Les Pages de propriétés

Toutes les propriétés des boîtes de dialogue communes peuvent être définies lors de l'exécution. Mais Visual Basic dispose d'un moyen plus simple de les configurer lors de la phase de création.

Le contrôle Common Dialog contient une propriété nommée `(Personnalisé)`. Lorsqu'on clique sur les points de suspension de cette propriété dans la fenêtre Propriétés, Visual Basic affiche la boîte de dialogue Pages de propriétés (voir Figure 9.6).

Figure 9.6
La boîte de dialogue Pages de propriétés permet de définir les propriétés lors de la création.

Cette boîte de dialogue Pages de propriétés permet de définir facilement quelques propriétés initiales pour la boîte de dialogue. Il s'agit, en fait, des propriétés les plus importantes de chaque style de boîte de dialogue. Exemple : pour sélectionner par défaut dans la boîte de dialogue Police le corps 12 et le style Gras, il suffit de saisir `12` dans le champ FontSize, et de cocher la case `Bold`.

La boîte de dialogue Ouvrir

Le Tableau 9.3 présente les valeurs de `Flags` que vous devrez définir avant d'appliquer la méthode `ShowOpen`. La boîte de dialogue Ouvrir, reproduite en Figure 9.7, offre à l'utilisateur une interface standard pour sélectionner le fichier à ouvrir. Ce type de boîte de dialogue gère de façon adéquate les connexions réseau.

Les valeurs présentées au Tableau 9.3 s'appliquent également à la propriété `Flags` de la boîte de dialogue Enregistrer.

Tableau 9.3 : Valeurs de Flags pour les boîtes de dialogue Ouvrir et Enregistrer

Littéral nommé	*Valeur*	*Description*
`cdlOFNAllowMultiselect`	`&H200`	Autorise, dans la liste déroulante Nom de fichier, les sélections multiples. La propriété `FileName` renverra alors une chaîne contenant le nom des fichiers (séparés par des espaces).
`cdlOFNCreatePrompt`	`&H2000`	Invite l'utilisateur à créer un fichier. Si cette valeur est définie, `cdlOFNPathMustExist` et `cdlOFNFileMustExist` doivent l'être également.
`cdlOFNExplorer`	`&H80000`	Affiche la boîte de dialogue selon le modèle Explorateur Windows.
`cdlOFNExtensionDifferent`	`&H400`	Indique que l'extension du nom de fichier renvoyé est différente de l'extension spécifiée par la propriété `DefaultExt`. Cette valeur n'est pas définie si la propriété `DefaultExt` contient `Null`, si les extensions correspondent ou si le fichier n'a pas d'extension. Cette valeur peut être inspectée, une fois la boîte de dialogue fermée.
`cdlOFNFileMustExist`	`&H1000`	L'utilisateur ne peut entrer que des noms de fichiers existants. Si cette valeur est définie, et que l'utilisateur entre un nom de fichier incorrect, un avertissement s'affiche. Cette valeur définit automatiquement `cdlOFNPathMustExist`.
`cdlOFNHelpButton`	`&H10`	Affiche le bouton Aide.
`cdlOFNHideReadOnly`	`&H4`	Masque la case à cocher Lecture seule.

Tableau 9.3 : Valeurs de Flags pour les boîtes de dialogue Ouvrir et Enregistrer *(suite)*

Littéral nommé	*Valeur*	*Description*
`cdlOFNLongNames`	`&H200000`	Autorise les noms de fichier longs.
`cdlOFNNoChangeDir`	`&H8`	Interdit le changement de dossier.
`cdlOFNNoDereferenceLinks`	`&H100000`	Interdit le déréférencement des *liens shell* (ou *raccourcis*). Par défaut, la sélection d'un lien shell le déréférence automatiquement auprès du shell.
`cdlOFNNoLongNames`	`&H40000`	Interdit les noms de fichier longs.
`cdlOFNNoReadOnlyReturn`	`&H8000`	Spécifie que le fichier renvoyé n'aura pas l'attribut Lecture seule et ne sera pas protégé en écriture.
`cdlOFNNoValidate`	`&H100`	Autorise les caractères invalides dans le nom de fichier renvoyé.
`cdlOFNOverwritePrompt`	`&H2`	Si le fichier sélectionné dans la boîte de dialogue Enregistrer sous existe déjà, génère un avertissement. (L'utilisateur peut choisir d'écraser le fichier.)
`cdlOFNPathMustExist`	`&H800`	Interdit les chemins d'accès non valides. Si cette valeur est définie et que l'utilisateur entre un chemin d'accès non valide, un avertissement s'affiche.
`cdlOFNReadOnly`	`&H1`	Coche par défaut l'option Lecture seule. Après la fermeture de la boîte de dialogue, cette valeur indique l'état de la case à cocher Lecture seule.
`cdlOFNShareAware`	`&H4000`	Spécifie que les éventuelles erreurs de violation de partage seront ignorées.

Souvent, dans les boîtes de dialogue de manipulation de fichiers, un filtre est appliqué aux extensions, de sorte que seuls apparaissent les fichiers d'une extension donnée (par exemple, les fichiers répondant à la sélection joker `*.doc`). Bien sûr, rien n'empêche l'utilisateur de passer outre à ce filtre, en appliquant un autre filtre ou en tapant `*.*`, pour afficher tous les types de fichiers. Mais vous pouvez spécifier le filtre par défaut à l'aide de la propriété `Filter` :

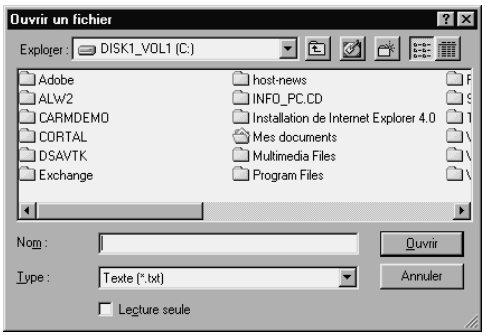


Figure 9.7
Une boîte de dialogue Ouvrir générée par le contrôle Common Dialog.

```
"FilterDescrip1 | extension1 | FilterDescrip2 | extension2 |
➥FilterDescrip3 | extension3"
```

Par exemple, l'instruction suivante applique un filtre selon lequel seuls les documents Word et Excel apparaîtront à l'affichage de la boîte de dialogue :

```
cdbFiles.Filter = "Documents Word (*.doc)|*.doc|Documents
➥Excel (*.xls)|*.xls"
```

Il ne faut pas confondre l'extension donnée dans la description et l'extension réelle spécifiée dans le filtre. En l'occurrence, `Word Docs (*.doc)` est ce qui s'affichera à l'utilisateur ; la première instruction de filtre est, en fait, le `*.doc` situé après la barre.

Vous pouvez spécifier un filtre multiple en incluant pour `Filter` plusieurs chaînes. Si vous spécifiez plusieurs filtres, vous devez affecter à la propriété `FilterIndex` le filtre à utiliser dans la boîte de dialogue Ouvrir courante. Pour le premier filtre, la valeur `FilterIndex` sera `1`, et ainsi de suite.

La propriété `FileName` du contrôle Common Dialog contient, après fermeture de la boîte de dialogue, le nom du fichier sélectionné.

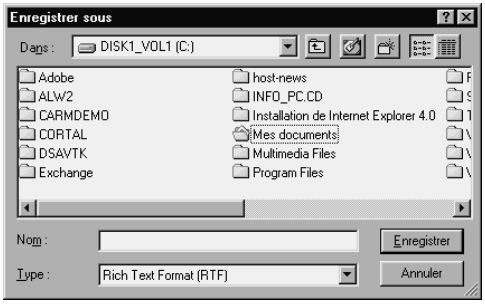
La boîte de dialogue Enregistrer

La boîte de dialogue Enregistrer est à peu près identique à la boîte de dialogue Ouvrir. Seuls changent de menus détails, tels les libellés de boutons de commande et quelques options. Si, par exemple, votre application exploite une interface multidocument (MDI), chaque document pouvant donc apparaître dans sa propre fenêtre, l'utilisateur pourra faire

une sélection multiple dans la boîte de dialogue Ouvrir. Cela est, par nature, impossible dans une boîte de dialogue Enregistrer.

La Figure 9.8 montre à quel point la boîte de dialogue Enregistrer ressemble à sa cousine Ouvrir. Les valeurs présentées au Tableau 9.3 s'appliquent aussi à la propriété `Flags` des boîtes de dialogue Enregistrer, incluant les filtres d'extensions.

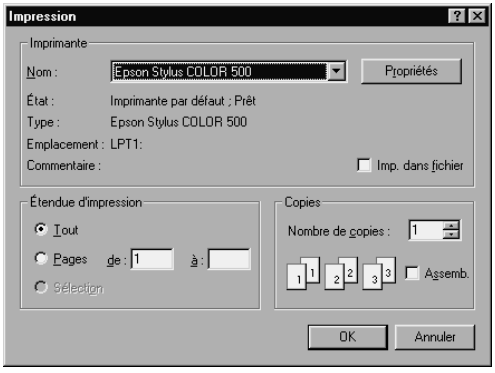
Figure 9.8
Une boîte de dialogue Enregistrer générée par le contrôle Common Dialog.



La boîte de dialogue Imprimer

La Figure 9.9 montre la boîte de dialogue Imprimer telle qu'elle s'affiche lorsque vous appliquez la méthode `ShowPrinter` à un contrôle Common Dialog. Dans cette boîte de dialogue, l'utilisateur peut sélectionner une imprimante, indiquer le nombre de copies, la séquence de pages à imprimer, etc. Le nombre et le type des options de la boîte de dialogue Imprimer dépend du pilote d'imprimante de l'utilisateur. Une fois que l'utilisateur a spécifié ses valeurs, votre programme interroge les propriétés du contrôle Common Dialog, et lance l'impression selon ces informations.

Figure 9.9
Une boîte de dialogue Imprimer générée par le contrôle Common Dialog.



Selon le type d'imprimante et de pilote, votre boîte de dialogue Imprimer peut différer de la Figure 9.9.

Le code du Listing 9.3 ouvre la boîte de dialogue Imprimer en réponse à une sélection de menu.

Listing 9.3 : Gestion de la boîte de dialogue Imprimer

```
1:  Private mnuFilePrint_Click()
2:     Dim intBegin As Integer, intEnd As Integer
3:     Dim intNumCopies As Integer, intI As Integer
4:     ' Suppose que Cancel est définie comme True.
5:
6:     On Error Goto dbErrHandler
7:     ' Affiche la boîte de dialogue Imprimer.
8:     cbdPrint.ShowPrinter
9:     ' Reçoit les valeurs sélectionnées par l'utilisateur.
10:    intBegin = cbdPrint.FromPage
11:    intEnd = cbdPrint.ToPage
12:    intNumCopies = cbdPrint.Copies
13:    '
14:    ' Imprime le nombre de copies demandé.
15:    For intI = 1 To intNumCopies
16:       ' Ici, code chargé de gérer la sortie imprimante.
17:    Next intI
18:    Exit Sub
19:
20: dbErrHandler:
21:    ' L'utilisateur a appuyé sur Annuler.
22:   Exit Sub
23: End Sub
```

Comme le montre le Listing 9.3, il n'y a pas de propriétés à définir avant de pouvoir afficher la boîte de dialogue Imprimer (sauf peut-être la propriété `DialogTitle`, qui spécifie le texte qui apparaîtra sur la barre de titre). Vous pouvez vérifier les valeurs renvoyées par la boîte de dialogue et stockées dans des propriétés comme `Copies`, `FromPage` ou `ToPage`. Ces valeurs correspondent aux sélections de l'utilisateur.

La boîte de dialogue Aide

Pour savoir comment intégrer l'aide Windows à vos programmes à partir du contrôle Common Dialog, il faudra patienter jusqu'au Chapitre 20.

En résumé

Vous êtes maintenant en mesure d'ajouter des boîtes de dialogue standards à vos programmes. Pour une boîte de dialogue d'ouverture de fichier, par exemple, il vaut mieux offrir à vos utilisateurs une interface familière. Vos applications n'en seront que plus conviviales.

Le contrôle Common Dialog implique que vous définissiez certaines propriétés, puis que vous lui appliquiez la méthode appropriée. Tout ce que fait ce contrôle, c'est de définir lui-même des propriétés. C'est à votre code de prendre le relais dès la fermeture de la boîte de dialogue, d'interpréter les sélections de l'utilisateur, et de gérer de façon adéquate le bouton Annuler.

Dans le prochain chapitre, vous apprendrez à suivre les mouvements de la souris afin de rendre vos applications réellement interactives. Vous apprendrez également à programmer les contrôles de type zone de liste, qui proposent diverses options à l'utilisateur.

Questions-réponses

Q Pourquoi le contrôle Common Dialog ne peut-il générer d'autres styles de boîtes de dialogue standards, tels que la boîte de dialogue Zoom de Word ou d'Excel ?

R Le contrôle Common Dialog ne peut pas tout faire. Il doit rester maniable et ne pas consommer trop de ressources mémoire. Si les programmes Windows les plus courants contiennent de multiples boîtes de dialogue, toutes ne sont pas assez communes pour constituer un véritable standard. Par exemple, la plupart des applications Windows se passent de la boîte de dialogue Zoom de Word et d'Excel. Du reste, les feuilles et contrôles Visual Basic vous permettent, avec un peu de travail, toutes les boîtes de dialogue que vous voulez.

Q Quels contrôles puis-je ajouter à ma Boîte à outils ?

R Vous pouvez enrichir de contrôles ActiveX votre collection d'outils. Cela inclut les contrôles ActiveX que vous écrivez vous-même (voir Chapitre 17), ainsi que les contrôles externes. Vous trouverez des contrôles de ce type sur le site Web de Microsoft, ainsi que sur beaucoup d'autres sites. Certaines revues spécialisées proposent aussi, parfois, des contrôles "prêts à l'emploi".

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront la mise en pratique de ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

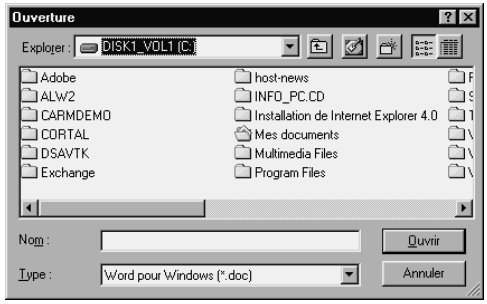
Quiz

1. Que faut-il faire dans la Boîte à outils avant de pouvoir placer un contrôle Common Dialog sur la feuille ?
2. Enumérez les différents styles de boîtes de dialogue que le contrôle Common Dialog peut générer.
3. A quoi sert le contrôle Common Dialog ?
4. Pourquoi ne peut-on redimensionner le contrôle Common Dialog sur la feuille ?
5. En fait, la boîte de dialogue Ouvrir n'ouvre aucun fichier. Vrai ou faux ?
6. A quoi sert la propriété `Filter` dans les boîtes de dialogue Ouvrir ou Enregistrer ?
7. Que fait la propriété `Flags` ?
8. Si aucune valeur de `Flags` n'est définie, Visual Basic ne peut afficher la boîte de dialogue Police. Vrai ou faux ?
9. Si aucune valeur de `Flags` n'est définie, Visual Basic ne peut afficher la boîte de dialogue Imprimer. Vrai ou faux ?
10. La méthode `Show` affiche un contrôle Common Dialog. Vrai ou faux ?

Exercices

1. Modifier le code du Listing 9.2 afin de gérer la sélection du bouton Annuler. Assurez-vous qu'aucune propriété ne sera changée si l'utilisateur clique sur Annuler.
2. Ecrivez une procédure qui génère la boîte de dialogue Ouvrir montrée à la Figure 9.10. Appliquez un filtre `*.txt`. Faites en sorte que les sélections de l'utilisateur soient annulées s'il clique sur le bouton Annuler.

Figure 9.10
Créez cette boîte de dialogue Ouvrir.



Chapitre 10

Gestion de la souris et contrôles avancés

Vous allez maintenant apprendre à écrire des applications qui réagissent à l'activité de la souris : déplacements, clics, glisser-déposer, etc. L'utilisation de la souris est inhérente à l'interface et aux applications Windows, et vos programmes doivent être en mesure de répondre d'une manière adéquate à ce type d'événements.

Nous explorerons aussi un nouveau genre de contrôles : les *contrôles de listes*. Vous avez, sans nul doute, déjà manipulé des zones de listes dans diverses applications. Ce chapitre vous explique comment créer et gérer vos propres contrôles de listes, et détaille les différents types proposés par Visual Basic.

Les contrôles de listes donnent à l'utilisateur le choix parmi plusieurs options prédéfinies. Ces contrôles nous amèneront également à étudier les *tableaux de variables*. Les tableaux de variables permettront à vos applications de traiter efficacement de grandes quantités de données.

Voici ce que nous découvrirons aujourd'hui :

- les événements souris ;
- comment identifier les clics de souris ;
- les opérations de glisser-déposer ;
- le contrôle timer ;

- les contrôles ListBox et ComboBox ;
- comment initialiser, ajouter ou supprimer des éléments de contrôles de listes ;
- les tableaux de variables ;
- les tableaux de contrôles.

Réponse à la souris

La réponse à la souris est un fondement des applications Windows. Si l'utilisateur se sert de sa souris lors de l'exécution, Windows passe les événements souris à votre programme. Les programmes que vous écrivez doivent interroger les événements souris et y répondre au moment et de la façon nécessaires. Si l'utilisateur se contente de cliquer sur un bouton d'option ou sur une case à cocher, naturellement, le programme n'a pas à répondre, car le contrôle déclenche un événement `Click`. Mais Visual Basic suit également les mouvements de la souris lorsque l'utilisateur effectue des glisser-déposer ou des copier-coller.

En fait, les programmes Windows doivent répondre aussi bien à la souris qu'au clavier. Le standard Windows veut que tous les programmes puissent être, si nécessaire, manipulables à partir du seul clavier. Ainsi, l'utilisateur qui préfère le clavier ou dont la souris est hors service pourra, malgré tout, exploiter ses applications. Il faut toutefois noter que certains programmes, par nature, se passent difficilement de la souris. Par exemple, un programme de dessin serait inimaginable sans le support de la souris.

La Boîte à outils ne contient aucun contrôle lié à la souris. Un programme répond à la souris par le biais des événements, et non par les propriétés de contrôles.

Les événements souris

Le comportement du programme par rapport aux événements souris est entièrement paramétrable. Un événement souris peut être déclenché par les actions suivantes :

- mouvement de la souris ;
- clic simple ;

- double-clic ;
- clic du bouton droit ;
- glisser-déposer.

Ajustement du curseur

Le curseur, ou pointeur, reflète à l'écran les mouvements physiques de la souris sur le tapis. La forme du pointeur (par défaut, une flèche) est souvent modifiée par l'application. Par exemple, lors d'une opération de glisser-déposer, ou lorsque l'utilisateur place le pointeur sur un objet indisponible. Le pointeur peut encore se transformer en sablier, pour indiquer qu'un traitement quelconque (tri de données, etc.) est en cours.

Votre application peut contrôler la forme du pointeur. Le Tableau 10.1 présente les différents types de pointeurs disponibles. Pour changer la forme du pointeur lorsqu'il passe sur un contrôle de la feuille, il faut définir la propriété `MousePointer` de ce contrôle. A peu près tous les contrôles disposent d'une propriété `MousePointer`, laquelle peut prendre l'une des valeurs présentées au Tableau 10.1. Ces valeurs sont à spécifier en cours d'exécution, par l'affectation de constantes nommées, ou lors de la création, par le réglage de la propriété `MousePointer` des contrôles.

Tableau 10.1 : La forme du pointeur est modifiable

Constante nommée	*Description*
`VbArrow`	Pointeur normal (flèche)
`VbCrosshair`	Pointeur cruciforme
`VbIbeam`	Curseur de texte
`VbIconPointer`	Petit carré dans un carré plus grand
`VbSizePointer`	Flèche à quatre pointes (haut, bas, gauche, droite)
`VbSizeNESW`	Flèche double (nord-est et sud-ouest)
`VbSizeNS`	Flèche double (haut et bas)
`VbSizeNWSE`	Flèche double (nord-ouest et sud-est)
`VbSizeWE`	Flèche double (droite et gauche)
`VbUpArrow`	Flèche vers le haut
`VbHourglass`	Sablier (attente)

Tableau 10.1 : La forme du pointeur est modifiable *(suite)*

Constante nommée	*Description*
`VbNoDrop`	Ne pas déposer (signe semblable au "stationnement interdit")
`VbArrowHourglass`	Flèche + sablier
`vbArrowQuestion`	Flèche + point d'interrogation
`vbSizeAll`	Double flèche de redimensionnement de fenêtre
`vbCustom`	Forme spécifiée par la propriété `MouseIcon`

Vous pouvez créer vos propres pointeurs. Le pointeur doit avoir une résolution de 16 × 16 pixels, comme les icônes. (Les fichiers d'icônes portent l'extension .ICO ; la plupart des programmes de dessin vous permettent de créer des icônes standards.) Pour afficher votre propre fichier icône à la place des pointeurs prédéfinis du Tableau 10.1, il suffit d'en affecter le chemin d'accès à la propriété `MouseIcon`*, et de définir la propriété* `MousePointer` *comme* `99 - Custom`*. La nouvelle forme restera le pointeur par défaut jusqu'à ce que vous en changiez de nouveau. Entre ce chapitre et le suivant, le Bonus projet 5, "Pratique de la souris,", vous en dira plus sur les pointeurs personnalisés.*

Déplacements et clics

Lorsque l'utilisateur déplace la souris ou clique, Windows génère des événements souris et les envoie à votre programme. Si le programme ne contient pas de procédures événementielles appropriées, ces événements seront ignorés. Il faut donc que le code pare à toute éventualité. Le Tableau 10.2 décrit chacun des événements souris.

Tableau 10.2 : Evénements souris générés par Windows

Evénement	*Description*
`Click`	L'utilisateur a cliqué sur un bouton de la souris
`DblClick`	L'utilisateur a double-cliqué sur un bouton de la souris
`MouseDown`	L'utilisateur a cliqué et maintient le bouton enfoncé
`MouseMove`	L'utilisateur a déplacé la souris
`MouseUp`	L'utilisateur a relâché le bouton de la souris

Les événements souris sont tous associés à des contrôles. En consultant la liste Objet de la fenêtre Code, vous constaterez qu'il existe, de même, des événements souris pour presque tous les contrôles, ainsi que pour les feuilles. Par exemple, pour répondre à un clic sur votre feuille `frmTest`, vous utiliseriez une procédure événementielle `frmTest_Click()`.

Certains événements liés aux clics de souris impliquent que vous interrogiez un argument de procédure événementielle pour déterminer sur quel bouton l'utilisateur a cliqué. Ces arguments ne sont passés que par les procédures événementielles `MouseDown` et `MouseUp`.

Un double-clic, est-ce un seul événement, ou deux événements `Click` ? En fait, la réponse dépend de la vitesse de l'utilisateur. Windows fournit les événements de clics dans l'ordre suivant :

1. `MouseDown`
2. `MouseUp`
3. `Click`
4. `DblClick`
5. `MouseUp`

Ainsi, l'événement `MouseDown` a lieu en premier lorsque l'utilisateur clique sur un bouton. Ensuite vient l'événement `MouseUp`, puis l'événement `Click`. Enfin, les événements `DblClick` et `MouseUp` se produisent si l'utilisateur double-clique. (Windows ne déclenche pas d'événement `MouseDown` quand l'utilisateur double-clique.)

Les procédures événementielles `MouseDown`, `MouseMove` et `MouseUp` exigent toujours ces quatre arguments :

- `intButton`. Indique de quel bouton il s'agit : `1` pour le bouton gauche, `2` pour le bouton droit, et `4` pour les deux (ou pour le bouton central sur les souris à trois boutons).
- `intShift`. Indique, à l'aide d'une comparaison de bits, si l'utilisateur a appuyé sur Alt, Ctrl ou Maj pendant le déplacement de la souris ou le clic.
- `sngX`. Coordonnée horizontale en twips correspondant au déplacement ou au clic.
- `sngY`. Coordonnée verticale en twips correspondant au déplacement ou au clic.

Visual Basic génère un événement de mouvement pour chaque déplacement de 10 à 15 twips, ce qui représente une infime portion de l'écran. (Visual Basic ne réagit pas à tous les twips.)

L'instruction suivante déclare une procédure événementielle `MouseDown` de sorte que l'on voie l'ordre de transmission des arguments :

```
 Private Sub imgMouse_MouseDown(intButton As Integer, intShift As
➥Integer, sngX As Single,
 [ic:ccc]sngY As Single)
```

`sngX` et `sngY` contiennent les coordonnées en twips du clic. `intButton` contient `1`, `2` ou `4`, selon le bouton sur lequel l'utilisateur a cliqué. Il n'est pas toujours indispensable de savoir quel bouton a été utilisé. Si le programme doit répondre différemment aux clics droits et aux clics gauches, on interrogera l'événement `MouseDown`. Pour savoir si l'utilisateur a conjointement appuyé sur Maj, Ctrl ou Alt, on procède à une vérification comme celle qui est mise en œuvre dans le Listing 10.1.

Listing 10.1 : Déterminer quelles touches ont été frappées conjointement à l'événement souris

```
1:  Private Sub imgMouse_MouseDown(intButton As Integer, intShift
    ➥As Integer, sngX As Single, sngY As Single)
2:    Dim intShiftState As Integer
3:    intShiftState = intShift And 7  ' And binaire.
4:    Select Case intShiftState
5:      Case 1
6:        ' Combinaisons Maj.
7:      Case 2
8:        ' Combinaisons Ctrl.
9:      Case 3
10:       ' Combinaisons Alt.
11:     Case 4
12:       ' Combinaisons Maj-Ctrl.
13:     Case 5
14:       ' Combinaisons Maj-Alt.
15:     Case 6
16:       ' Combinaisons Ctrl-Alt.
17:     Case 7
18:       ' Combinaisons Maj-Ctrl-Alt.
19:     End Select
20: End Sub
```

A la ligne 3, la comparaison `And` spéciale interroge un bit interne pour déterminer les touches conjointement utilisées.

Faire

Interrogez les combinaisons touches/souris lorsque vos applications permettent la sélection de texte avec la touche Maj, ou la sélection multiple avec la touche Ctrl. Beaucoup de contrôles, tels que les zones de listes, que vous découvrirez plus loin, gèrent automatiquement les combinaisons Ctrl-souris

pour la sélection des éléments. Pour ces contrôles, vous n'avez donc pas à vous occuper de ce type d'événements clavier/souris.

Le Projet bonus 5, situé entre ce chapitre et le suivant, contient le code d'une application complète qui met en œuvre la gestion des mouvements et des clics de souris.

Les opérations de glisser-déposer

Votre application doit également, si nécessaire, être en mesure de suivre les opérations de glisser-déposer. Le glisser-déposer est l'opération pendant laquelle l'utilisateur clique sur un objet, maintient le bouton enfoncé, et fait glisser l'objet vers un autre emplacement. La programmation du glisser-déposer est fort simple, car le système d'exploitation Windows envoie tout au long de l'opération les informations adéquates.

Visual Basic supporte deux types d'opérations de glisser-déposer :

- le glisser-déposer automatique ;
- le glisser-déposer manuel.

La première méthode est la plus simple. Le glisser-déposer automatiquement se gère à l'aide de propriétés de contrôles. A peu près tous les contrôles de la Boîte à outils disposent de la propriété `DragMode`. Cette propriété permet à l'utilisateur de déplacer le contrôle avec la souris. Au cours du déplacement, Visual Basic affiche la silhouette du contrôle. Il reste à s'assurer que le contrôle arrive bien à destination, là où l'utilisateur relâche le bouton. Si le glisser-déposer automatique illustre le déplacement, il ne déplace pas réellement l'objet.

L'événement `DragDrop` de la feuille détermine le point de chute de l'objet. Pour configurer le glisser, il suffit de définir la propriété `DragMode` du contrôle comme `1 - Automatic`. Le contrôle peut alors être déplacé par glissement, et la silhouette suivra le déplacement. La procédure événementielle `Form_DragDrop()` prend en charge la suite de l'opération, et place le contrôle à l'endroit voulu.

Par défaut, c'est la silhouette du contrôle qui apparaît lors du déplacement. Vous pouvez remplacer cette silhouette par une icône spécifique, en affectant à la propriété `DragIcon` le chemin d'accès d'un quelconque fichier icône (par exemple, les fichiers fournis avec Visual Basic dans le dossier \Graphics). Cette icône apparaîtra en lieu et place de la silhouette réelle lors du glisser-déposer. Quand un utilisateur fera glisser le contrôle, l'icône se substituera au pointeur. Voici un exemple de code gérant le glisser-déposer :

```
1:  Private Sub frmTitle_DragDrop(Source As Control, X As Single, Y As Single)
2:  ' Ici, le contrôle déplacé
3:  ' est passé comme argument.
4:    Source.Move X, Y  ' Déplace le contrôle.
5:  End Sub
```

A la ligne 4, la méthode `Move` applique au contrôle les coordonnées de son nouvel emplacement (là où l'utilisateur a relâché le bouton).

L'événement `DragOver` a lieu lorsque l'utilisateur fait glisser un contrôle sur un autre. Pour indiquer qu'une telle opération n'est pas permise par le contrôle de destination, vous pouvez modifier l'icône. Il suffit, pour cela, de changer le pointeur dans les procédures événementielles `DragOver` des contrôles qui n'acceptent pas qu'on dépose un objet dessus. `DragOver` peut recevoir quatre arguments :

- *le contrôle ;*
- *les coordonnées x du pointeur ;*
- *les coordonnées y du pointeur ;*
- *l'état du déplacement, qui prend trois valeurs différentes : `0` (le déplacement atteint l'objet), `1` (le déplacement quitte l'objet), et `2` (le contrôle est déplacé au-dessus de l'objet).*

Le glisser-déposer manuel fonctionne comme le glisser-déposer automatique, à trois différences près :

- La propriété `DragMode` doit être définie comme `0 - Manual`.
- Le contrôle répond à un événement `MouseDown` avant le début du déplacement, de sorte que les coordonnées initiales du contrôle soient enregistrées.
- Le code est ajouté dans la procédure événementielle `MouseDown`.

Pour achever le glisser-déposer, la procédure événementielle `MouseDown` applique à l'objet la méthode `Drag`. Le code suivant déplace l'image à condition que la propriété `DragMode` du contrôle image soit définie comme `0 - Manual` :

```
Private Sub imgMouse_MouseDown(Button As Integer, Shift As
➥Integer, X As Single, Y As Single)
  ' On a cliqué sur l'image.
  txtMouse.Text = "On a cliqué sur l'image à la position " & X & "," & Y
  imgMouse.Drag
End Sub
```

La méthode `Drag` autorise le glisser-déposer. Sans la méthode `Drag`, la procédure événementielle `MouseDown()` ne peut gérer l'opération. On se sert du glisser-déposer manuel pour imposer des restrictions au déplacement, avant et pendant l'opération.

Le Projet bonus 7 situé après le Chapitre 14, propose une application qui met en œuvre les clics du bouton droit et les menus contextuels.

Les contrôles *ListBox*

Vous allez maintenant découvrir les autres contrôles qui apparaissent dans la Boîte à outils. Les contrôles que nous allons étudier exigent un peu de programmation pour fonctionner. Il ne suffit pas, comme pour les boutons de commande, de les disposer sur la feuille ; il faut aussi les initialiser, ce qui n'est faisable qu'à partir du code. Par exemple, vous ne pouvez pas remplir une zone de liste déroulante de ses éléments lors de la phase de création. Les contrôles multivaleurs ne s'initialisent pas aussi facilement que les labels. Pour cela, il faut programmer.

Voici les différents types de zones de liste :

- zone de liste simple ;
- ComboBox liste déroulante ;
- ComboBox simple ;
- ComboBox déroulante.

Les zones de liste simples

Le contrôle zone de liste simple permet à l'utilisateur de sélectionner un ou plusieurs éléments dans une liste prédéfinie. Pour ajouter une zone de liste simple à votre feuille, double-cliquez sur le contrôle ListBox de la Boîte à outils.

On peut initialiser depuis la fenêtre Propriétés certaines propriétés des zones de liste : position, taille, couleur, etc. Il n'en va pas de même pour la liste des valeurs du contrôle.

Si vous n'initialiserez pas, en général, la liste de valeurs du contrôle ListBox depuis la fenêtre Propriétés, c'est que Visual Basic vous permet de le faire lors de l'exécution. Dans la fenêtre Propriétés, cliquez sur la propriété `List` *d'un contrôle ListBox : une liste s'affiche, dans laquelle vous pouvez ajouter des valeurs. Mais les valeurs des contrôles ListBox, la plupart du temps, viennent de l'utilisateur ou d'autres sources de données. Seules devraient être initialisées depuis la fenêtre Propriétés les petites zones de liste dont les valeurs ne sont pas appelées à changer. Pour le reste, vous utiliserez des méthodes qui construisent la liste au cours de l'exécution, ce qu'explique la suite de ce chapitre.*

Pour ajouter des éléments à une liste, on se sert de la méthode `AddItem` (voir Listing 10.2).

Listing 10.2 : La procédure événementielle initialise le contrôle ListBox

```
1:  Private Sub Form_Load()
2:    ' Initialise les valeurs du contrôle.
3:    lstColors.AddItem "Rouge"
4:    lstColors.AddItem "Bleu"
5:    lstColors.AddItem "Vert"
6:    lstColors.AddItem "Jaune"
7:    lstColors.AddItem "Orange"
8:    lstColors.AddItem "Blanc"
9:  End Sub
```

La procédure événementielle `Form_Load()` charge les valeurs initiales. Naturellement, le plus gros des valeurs sera ajouté lors de l'exécution. Grâce à la procédure événementielle `Form_Load()`, la partie disponible des valeurs sera déjà chargée lorsque l'utilisateur affichera la feuille contenant la zone de liste. Au démarrage du programme, le module d'exécution charge les feuilles. Le chargement des feuilles déclenche l'événement `Load`. Les commandes `Load` et `Unload` permettent de spécifier à quel moment précis une feuille doit être chargée, ou déchargée pour libérer des ressources.

Après le chargement des valeurs initiales, c'est la méthode `AddItem` qui permettra d'ajouter les autres éléments en cours d'exécution. Le Listing 10.2 donnait en exemple l'application de `AddItem` à la zone de liste nommée `lstColors`. Il suffit de spécifier l'élément après le nom de la méthode. Les éléments seront ajoutés à la liste dans l'ordre selon lequel vous les avez entrés — à moins que vous ne définissiez la propriété `Sorted` comme `True`, auquel cas les éléments apparaîtront dans l'ordre alphabétique ou numérique.

L'initialisation des valeurs de la propriété `List` depuis la fenêtre Propriétés rend la maintenance moins aisée. Si les valeurs initiales sont spécifiées dans le code, il sera plus simple d'ajouter des éléments par la suite.

L'instruction suivante ajoute la couleur Aqua à la liste configurée dans la procédure événementielle `Form_Load()` :

```
lstColors.AddItem "Aqua"
```

Il n'est pas toujours évident de distinguer commandes, déclarations, méthodes, propriétés et contrôles. Considérez la méthode comme une sorte de requête que l'objet fait pour lui-même. Ainsi, `lstColors.AddItem "Aqua"` ne signifie rien d'autre que : "Ajoute-moi un élément nommé Aqua". En tant que contrôle ListBox, `lstColors` sait comment répondre à cette requête, parce que la méthode fait partie de son répertoire. La plupart des contrôles disposent de

leur propre jeu de méthodes. A propos des contrôles externes que vous pourrez ajouter, la liste des méthodes est généralement fournie dans la documentation.

La procédure `Form_Load()` ci-dessus charge une feuille avec, en son centre, la zone de liste `lstColors`. (Voir Figure 10.1.)

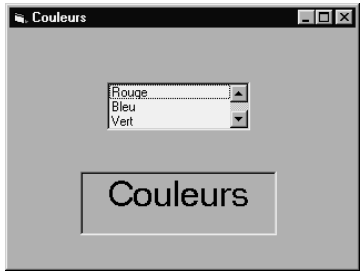


Figure 10.1
La zone de liste contient les éléments spécifiés.

Si vous initialisez des valeurs de la zone de liste lors de la création, il faut vider la propriété `List` dans la fenêtre Propriétés après avoir nommé le contrôle. Visual Basic affiche automatiquement le nom du contrôle ListBox comme premier élément de la liste.

La Figure 10.1 montre une *zone de liste simple*.

Une zone de liste simple affiche les éléments spécifiés dans la liste des valeurs. C'est le programmeur qui initialise la liste ; l'utilisateur ne peut ajouter directement des éléments.

Si la hauteur ou la largeur de la zone de liste ne permet pas d'afficher tous les éléments, des barres de défilement verticale et horizontale apparaissent. Et si des valeurs viennent s'ajouter à la liste dans le cours de l'exécution, Visual Basic ajoute automatiquement les barres de défilement.

Le but de la zone de liste est de permettre à l'utilisateur de choisir dans une liste plutôt que d'avoir à saisir les valeurs. Lorsque l'utilisateur sélectionne un élément de la liste, voici ce qui se passe :

- L'élément sélectionné est mis en surbrillance.
- La sélection est copiée dans la propriété `Text` de la zone de liste. Ainsi, `lstColors.Text` change au cours de l'exécution, selon les choix de l'utilisateur.

`Text` ne peut contenir qu'une seule valeur. Pour mieux illustrer cette propriété, imaginons qu'une zone de texte nommée `txtColor` soit ajoutée à la feuille de la Figure 10.1.

Dès que l'utilisateur fait une sélection dans `lstColors`, la procédure suivante envoie l'élément à la zone de texte :

```
Private Sub lstColors_Click()
  ' Copie l'élément sélectionné dans la zone de texte.
  txtColor.Text = lstColors.Text
End Sub
```

Si vous placez sur la feuille une zone de texte destinée à afficher l'élément sélectionné, il faut vider la propriété `Text` de la zone de texte dans la fenêtre Propriétés. De la sorte, rien n'apparaîtra avant que l'utilisateur n'ait effectué une sélection. La propriété `Font` des zones de texte doit presque toujours être modifiée, car la valeur par défaut donne un affichage trop petit. Pour afficher plus d'un élément à la fois, il peut aussi être nécessaire d'agrandir la zone de liste. Dans la Figure 10.2, l'élément sélectionné dans la zone de liste a été envoyé à la zone de texte.

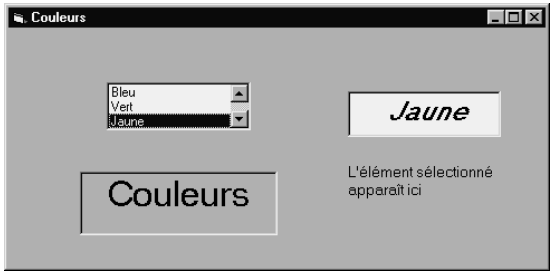


Figure 10.2
La zone de texte contient l'élément de liste sélectionné.

Chaque élément de la zone de liste se voit affecter une valeur index, pour être distingué des autres. L'index commence à `0` pour le premier élément, et ainsi de suite (il ne peut y avoir de doublons). La propriété `ListIndex` de la zone de liste contient la valeur d'index de l'élément sélectionné. Vous pouvez ainsi déterminer quel élément a été choisi. D'autres méthodes du contrôle ListBox, telles que `RemoveItem`, se servent des valeurs de la propriété `ListIndex`. Par exemple, pour retirer le troisième élément de la liste, voici ce que serait le code :

```
lstColors.RemoveItem 2
```

A chaque suppression d'un élément, les valeurs d'index sont automatiquement renumérotées. Exemple : le quatrième élément de la liste a la valeur d'index `3` (rappelons que l'index commence à `0`), mais si vous supprimez un élément situé avant, le quatrième élément deviendra le troisième, et prendra `2` pour valeur d'index. Si la propriété `Sorted` est définie comme `True`, Visual Basic retrie les valeurs d'index après chaque ajout ou suppression.

Vous pouvez, si nécessaire, permettre à l'utilisateur de faire des sélections multiples dans la zone de liste. L'utilisateur appuie sur la touche Ctrl et clique sur divers éléments. Mais la propriété `Value` *de la zone de liste ne peut contenir toutes les valeurs sélectionnées. Que faire ? Vous l'apprendrez dans le Projet bonus 5, à la suite du Chapitre 11.*

Les valeurs d'index des zones de liste ressemblent beaucoup aux tableaux, que nous étudierons à la fin de ce chapitre.

L'utilisateur ne peut lui-même ajouter des valeurs à la zone de liste. Cela, seul le code peut le faire, à l'aide de la méthode `AddItem`. L'utilisateur ne peut, non plus, supprimer des éléments. C'est le privilège de la méthode `RemoveItem`.

Supposons qu'il faille retirer tous les éléments que nous avons ajoutés à la zone de liste. On pourrait appliquer la méthode `RemoveItem` à chacun. Mais il y a plus simple : une boucle `For`. Le code suivant reprend l'exemple de la zone de liste `lstColors` :

```
1:  ' Supprime les cinq éléments de la liste.
2:  For intI = 0 To 5
3:    lstColors.RemoveItem 0
4:  Next intI
```

Naturellement, pour vider entièrement la liste à l'aide d'une boucle `For`, il faut connaître le nombre exact des éléments. Mais, chaque élément ayant été ajouté dans le code, cela ne devrait pas être bien difficile.

On peut aussi permettre à l'utilisateur d'ajouter des valeurs à la liste, à l'aide, par exemple, d'une zone de texte et d'un bouton de commande qui déclencherait la méthode `AddItem`. Même dans un tel cas, il reste possible de tenir une comptabilité précise des valeurs de la liste, via une variable qui serait incrémentée ou décrémentée chaque fois que l'utilisateur ajoute ou supprime un élément.

En réalité, il est inutile de recourir à une variable compteur de ce type. Visual Basic se charge, en interne, de suivre l'évolution du contenu de la liste. La propriété `ListCount` contient et met à jour le nombre total des éléments de la liste. La valeur de `ListCount` est toujours supérieure à la plus grande valeur de `ListIndex`, puisque `ListIndex` commence à `0`.

Ainsi, pour connaître le nombre actuel des éléments de la liste, ou pour leur appliquer une méthode à l'intérieur d'une boucle `For`, la propriété `ListCount` peut être utilisée comme ceci :

```
1:  ' Supprime tous les éléments de la liste.
2:  intTotal = lstColors.ListCount  ' Stocke le nombre.
3:  For intI = 1 To intTotal
4:    lstColors.RemoveItem 0
5:  Next intI
```

Le contrôle ListBox dispose d'une méthode, `Clear`, qui supprime instantanément tous les éléments, sans qu'une boucle soit nécessaire. Si donc vous voulez vider complètement la liste, l'instruction `lstColors.Clear` est ce qui se fait de plus simple.

Les contrôles *ComboBox*

Le contrôle ComboBox est disponible en trois parfums :

- ComboBox liste déroulante ;
- ComboBox simple ;
- ComboBox déroulante.

Alors que la zone de liste apparaît toujours dans la forme et la taille que vous lui avez attribuées, la ComboBox liste déroulante (ou zone de liste déroulante modifiable) s'affiche sur une seule ligne jusqu'à ce que l'utilisateur clique dessus pour l'ouvrir.

La ComboBox simple est une zone de liste simple combinée à une zone de texte. L'utilisateur peut ajouter des éléments à la liste en entrant des valeurs dans la zone de texte connexe.

La ComboBox déroulante (ou zone de liste déroulante fixe) n'occupe sur la feuille qu'une seule ligne, jusqu'à ce que l'utilisateur sélectionne le contrôle. Alors seulement, la ComboBox s'ouvre et affiche la liste des éléments. L'utilisateur peut sélectionner un élément ou entrer une nouvelle valeur dans la zone de texte connexe.

Pour ce qui est de l'initialisation et de la manipulation à partir du code, les contrôles ComboBox fonctionnent comme les zones de liste. La différence principale tient à l'aspect sur la feuille et à la façon dont l'utilisateur y opère ses sélections.

Les trois sortes de ComboBox s'obtiennent à partir du même contrôle ComboBox de la Boîte à outils. Après avoir placé le contrôle sur la feuille, vous en spécifiez le type à l'aide de la propriété `Style`. Le style par défaut est `0 - Dropdown Combo` (ComboBox déroulante).

Rappelez-vous que la propriété `Sorted` trie automatiquement les éléments de la liste, selon l'ordre alphabétique ou numérique ; et cela, même si l'utilisateur ajoute des valeurs. Si la propriété `Sorted` n'est pas définie comme `True`, les éléments apparaîtront dans l'ordre dans lequel ils ont été ajoutés.

La ComboBox liste déroulante fonctionne comme une zone de liste, à cela près que cette dernière occupe généralement plus de place sur la feuille. La ComboBox liste déroulante ne s'ouvre pour afficher son contenu que lorsque l'utilisateur clique sur la petite flèche.

La Figure 10.3 montre une feuille présentant deux ComboBox liste déroulante, qui contiennent les mêmes éléments. Avant que celle de droite n'ait été déroulée, les deux ComboBox occupaient le même espace sur la feuille.

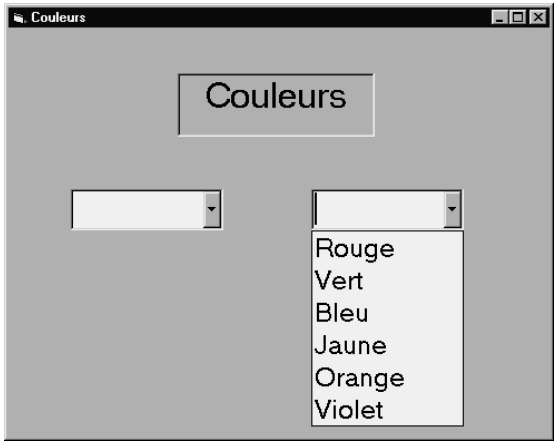


Figure 10.3
Avant que l'utilisateur ne la sélectionne, la ComboBox liste déroulante occupe très peu d'espace.

On se sert des ComboBox liste déroulante pour offrir à l'utilisateur de multiples choix tout en économisant de l'espace sur la feuille. Comme vous pouvez le voir sur la Figure 10.3, la ComboBox liste déroulante n'affiche pas de texte tant que l'utilisateur ne l'a pas sélectionnée. Il convient donc d'indiquer à l'utilisateur, *via* un label ou une boîte de message, quels types d'éléments lui sont proposés.

On ajoute des entrées aux ComboBox liste déroulante comme aux zones de liste : par la méthode `AddItem`. Le code suivant ajoute six couleurs aux contrôles de la Figure 10.3 :

```
Private Sub Form_Load()
' Initialise les deux ComboBox.
    cboColor1.AddItem "Rouge"
    cboColor1.AddItem "Bleu"
    cboColor1.AddItem "Vert"
    cboColor1.AddItem "Jaune"
    cboColor1.AddItem "Orange"
    cboColor1.AddItem "Blanc"
    '
    cboColor2.AddItem "Rouge"
```

```
    cboColor2.AddItem "Bleu"
    cboColor2.AddItem "Vert"
    cboColor2.AddItem "Jaune"
    cboColor2.AddItem "Orange"
    cboColor2.AddItem "Blanc"
End Sub
```

Les ComboBox liste déroulante supportent les mêmes méthodes et les mêmes propriétés que les zones de liste.

Le deuxième style de Comb oBox, la ComboBox simple, fonctionne comme une combinaison de zone de liste et de zone de texte. Ainsi, l'utilisateur peut sélectionner un élément ou en entrer un nouveau. Lorsque la propriété `Style` est définie comme `1 - Simple Combo`, le contrôle peut être dimensionné comme une zone de liste. Si, à un moment donné de l'exécution, la hauteur et la largeur spécifiées ne permettent pas d'afficher tout le contenu, des barres de défilement s'affichent automatiquement.

Sur la Figure 10.4, la liste des couleurs est contenue dans une ComboBox simple. Les éléments sont ajoutés à l'aide de la méthode `AddItem` ; les utilisateurs peuvent aussi entrer de nouvelles valeurs. Après avoir disposé le contrôle sur la feuille, vous devez en effacer la propriété `Text`. Autrement, le nom du contrôle apparaîtrait dans le champ de la zone de texte. Vous pouvez aussi bien spécifier dans cette propriété `Text` une valeur par défaut, que l'utilisateur sera libre d'accepter ou de modifier.

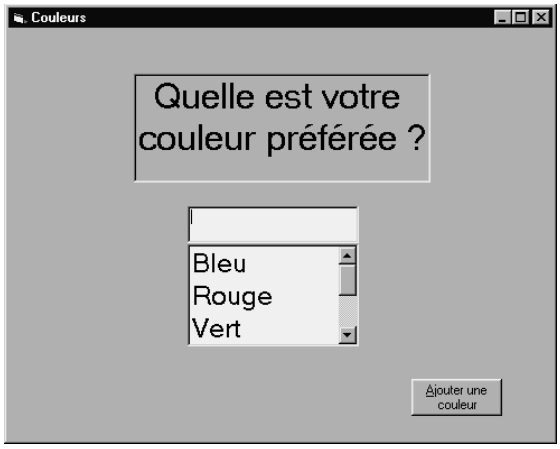


Figure 10.4
L'utilisateur peut sélectionner une couleurs ou entrer une nouvelle valeur.

Dans la ComboBox simple de la Figure 10.4, les couleurs apparaissent dans l'ordre alphabétique. Vous avez deviné ? Eh oui, la propriété `Sorted` est définie comme `True`.

Si la propriété `Style` de la ComboBox simple est définie comme `1 - Simple Combo`, vous devez spécifier les dimensions du contrôle. A la différence des listes déroulantes, les ComboBox simples gardent la taille qui leur a été attribuée.

La ComboBox simple n'ajoute pas automatiquement les entrées utilisateur. C'est à vous et à votre code de faire en sorte que le contenu soit mis à jour — si, du moins, l'utilisateur est autorisé à entrer de nouvelles valeurs. Les ComboBox simples ne servent généralement qu'à proposer à l'utilisateur des choix prédéfinis. Pour lui permettre d'ajouter des éléments, vous devez écrire pour le contrôle une procédure événementielle `LostFocus()`, comme celle-ci :

```
Private Sub cboColor_LostFocus()
  cboColor2.AddItem cboColor1.Text
End Sub
```

La procédure événementielle `LostFocus()` s'exécute lorsque le contrôle perd le focus, ce qui se produit quand l'utilisateur clique sur un autre contrôle, ou déplace le focus en appuyant sur Tab. Dès que le focus passe à un autre contrôle, la procédure événementielle `LostFocus()` s'exécute, et la valeur entrée par l'utilisateur dans la ComboBox (valeur stockée dans la propriété `Text`) est affectée à l'aide de la méthode `AddItem`. La feuille de la Figure 10.4 contenait un bouton de commande, car le focus doit passer à un autre contrôle pour que la valeur saisie par l'utilisateur soit enregistrée. C'est à cette seule condition que de nouvelles entrées sont ajoutées aux ComboBox simples.

La ComboBox déroulante est, sans doute, la plus intéressante. Le contrôle reste fermé jusqu'à ce que l'utilisateur le sélectionne ; ce qui économise de la place. Dans la ComboBox déroulée, l'utilisateur peut sélectionner une valeur existante ou ajouter un nouvel élément. `Selected`, `ListCount`, ainsi que d'autres propriétés des zones de liste qui s'appliquent également aux ComboBox déroulantes. Contrairement aux zones de liste déroulante, les ComboBox déroulantes autorisent l'ajout de nouveaux éléments.

Assurez-vous que la propriété `Style` du contrôle est bien définie comme `0 - Dropdown Combo` (valeur par défaut). La Figure 10.5 reprend l'exemple de la liste de couleurs — mais, cette fois, c'est une ComboBox déroulante qui est utilisée.

Prenez garde de bien vider la propriété `Text` du contrôle lorsque vous disposez votre ComboBox déroulante sur la feuille ; sans quoi, le nom du contrôle apparaîtrait dans la zone de texte. Comme pour la ComboBox simple, les valeurs éventuellement ajoutées par l'utilisateur ne sont stockées qu'à partir du moment où le contrôle perd le focus.

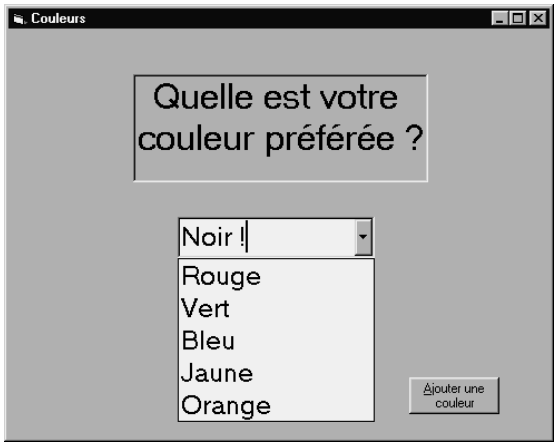


Figure 10.5
L'utilisateur peut sélectionner un élément ou entrer une nouvelle valeur.

Le contrôle Timer

Le contrôle Timer vous permet de générer des réponses sur la base des valeurs envoyées par l'horloge interne du PC. Vous pouvez ainsi écrire un code qui s'exécutera au bout d'un certain laps de temps, ou programmer des traitements en arrière-plan. L'horloge interne déclenche un événement timer dix-huit fois par seconde. Cette horloge est capitale pour le fonctionnement du processeur, de la mémoire et du disque dur, les données étant traitées selon un ordre très strict.

Votre PC génère un événement timer dix-huit fois par seconde, quelle que soit la vitesse du processeur.

Les applications Visual Basic répondent aux événements timer. Ces événements sont générés par l'horloge interne du PC, et Windows les envoie au programme en cours d'exécution. Vous pouvez prédéfinir l'intervalle de temps dans lequel Windows envoie cette information à votre programme. Comme pour les autres événements, vous pouvez aussi écrire des procédures événementielles qui s'exécuteront chaque fois qu'un événement timer aura lieu. Retenez ceci : le code peut s'exécuter sur la base d'intervalles de temps, indépendamment de la vitesse du PC, parce que le temps est une constante.

Le contrôle Timer reçoit les événements timer et y répond en fonction des propriétés que vous avez définies. Lorsque vous disposez le contrôle Timer sur la feuille, vous déterminez la fréquence des événements timer. Cet intervalle correspond à l'une des

propriétés du contrôle Timer. Lorsque ce laps de temps s'est écoulé, le contrôle Timer déclenche la procédure événementielle appropriée.

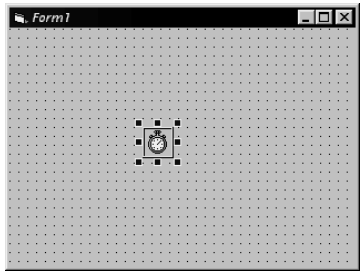
Comme pour les autres contrôles, vous pouvez placer sur la feuille autant de contrôles Timer que vous voulez. Par exemple, vous pouvez placer trois contrôles Timer différents, dont l'un déclenchera un événement spécifique toutes les minutes, l'autre toutes les demi-heures, et le troisième toutes les heures.

C'est en expérimentant que vous découvrirez tous les emplois possibles du contrôle Timer. Les procédures événementielles `Timer` *permettent, notamment, de piloter des tâches en arrière-plan. Il est aussi possible de simuler une animation en redessinant une image à chaque déclenchement de l'événement, par exemple toutes les demi-secondes.*

Pour disposer le contrôle Timer sur la feuille, double-cliquez sur l'outil correspondant de la Boîte à outils, puis placez le contrôle résultant à l'écart des autres contrôles de la feuille. Comme le contrôle Common Dialog, le contrôle Timer ne peut être dimensionné, et n'apparaît pas sur la feuille lors de l'exécution.

La Figure 10.6 montre un contrôle Timer au centre de la feuille.

Figure 10.6
Le contrôle Timer peut être déplacé sur la feuille, mais non redimensionné.



Le contrôle Timer supporte très peu de propriétés. Des six propriétés définissables lors de la création, cinq sont plutôt communes :

- `Left` et `Top` spécifient la position du contrôle.
- `Enabled` active le contrôle.
- `Tag` contient les informations que vous êtes libre d'adjoindre au contrôle.
- `Index` spécifie l'indice du contrôle dans un tableau de contrôle.

Si vous définissez comme `False` la propriété `Enabled` lors de la création, le contrôle Timer ne commencera à répondre aux événements que lorsque le code définira `Enabled` comme `True`.

La seule propriété réellement spécifique au contrôle Timer est `Interval`. La propriété `Interval` spécifie la fréquence à laquelle le contrôle répondra aux événements. Cet intervalle en millisecondes peut être défini lors de la création ou lors de l'exécution. Si, par exemple, vous affectez à la propriété `Interval` la valeur `1000`, les événements timer se produiront toutes les 1 000 millisecondes, soit une fois par seconde environ.

Evidemment, le contrôle Timer connaît quelques limites. Notamment, la propriété `Interval` ne peut contenir que des valeurs situées entre `0` et `64,767`. L'intervalle maximal sera donc de 65 secondes, pas plus. Il est cependant possible, dans la procédure événementielle `Timer` du contrôle, d'ignorer les événements jusqu'à ce qu'un certain laps de temps se soit écoulé. Ainsi, même si la procédure événementielle du contrôle se déclenche toutes les 60 secondes, vous pouvez faire en sorte que les événements ne rencontrent une réponse qu'une heure (ou deux, ou plus) après la dernière exécution de la procédure événementielle.

Le contrôle Timer n'est pas d'une exactitude absolue. Les cristaux à l'intérieur de l'ordinateur sont très précis, mais le temps que Windows envoie l'événement au programme, un peu de cette précision s'est perdu. En outre, d'autres événements système (transfert par réseau, activité du modem, etc.) peuvent ralentir l'activité du timer. Malgré les apparences, l'ordinateur ne peut réellement pas faire deux choses à la fois, et le contrôle Timer de votre application Visual Basic n'a pas toujours la priorité. Conclusion : le contrôle Timer est efficace pour des tâches réclamant une ponctualité à la seconde près. Visual Basic ne dispose pas de contrôles plus précis.

Le contrôle Timer ne supporte qu'un seul événement : l'événement `Timer`. Ainsi, pour un contrôle Timer nommé `tmrClock`, vous ne pourrez écrire qu'une seule procédure événementielle : `tmrClock_Timer()`. Cette procédure contiendra le code qui doit s'exécuter chaque fois que s'écoule l'intervalle spécifié.

A titre d'exercice, nous vous invitons à créer une application simple qui mette en œuvre le contrôle Timer. Suivez ces étapes :

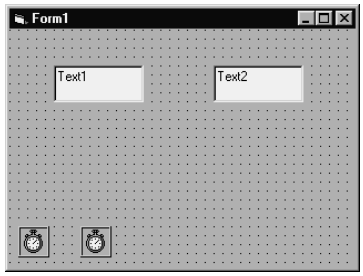
1. Créez un nouveau projet et placez un contrôle Timer sur la feuille.
2. Affectez à la propriété `Interval` la valeur `1000`, de sorte que le contrôle réponde à la procédure événementielle toutes les secondes.

3. Nommez le contrôle `tmrTimer1` et placez-le dans le coin inférieur gauche de la feuille.

4. Ajoutez, à côté du premier, un contrôle Timer nommé `tmrTimer2`. Affectez à sa propriété `Interval` la valeur `500`, de sorte que le contrôle réponde à la procédure événementielle toutes les demi-secondes.

5. Ajoutez à la feuille deux zones de texte nommées `txtTime1` et `txtTime2`. Disposez-les conformément à la Figure 10.7.

Figure 10.7
L'application timer est presque terminée.



6. Affectez la valeur `1` à la propriété `Text` de chaque zone de texte, et réglez la taille de la police à `18`. Définissez également les deux propriétés `Alignment` comme `2 - Center`, de sorte que le texte soit centré dans les zones de texte. Affectez enfin aux deux propriétés `Width` la valeur `1000`.

7. Double-cliquez sur le premier contrôle Timer. Dans la fenêtre Code qui s'affiche, entrez la procédure événementielle `tmrTime1_Timer()` suivante (Visual Basic insère automatiquement les lignes d'encadrement) :

```
Private Sub tmrTimer1_Timer()
  ' Ajoute 1 à la valeur affichée.
  txtTimer1.Text = txtTimer1.Text + 1
End Sub
```

8. Ajoutez au second timer la procédure événementielle suivante :

```
Private Sub tmrTimer2_Timer()
  ' Ajoute 1 à la valeur affichée.
  txtTimer2.Text = txtTimer2.Text + 1
End Sub
```

Ce code incrémente de 1 la valeur des zones de texte (ces valeurs sont par défaut de type `Variant`).

9. Ajoutez un label sous chaque zone de texte : "Lent" pour la première ; et "Rapide" pour la seconde.
10. Exécutez l'application. Votre feuille devrait ressembler à la Figure 10.8. Voici ce qui se produit : la première zone de texte s'actualise toutes les secondes, l'autre toutes les demi-secondes. Dans les limites de la précision du contrôle Timer, la seconde zone de texte devrait donc se mettre à jour deux fois plus rapidement que la première.

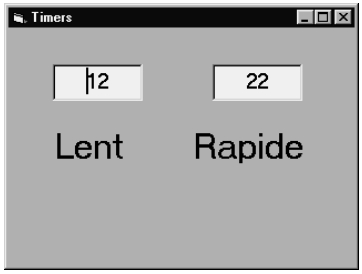


Figure 10.8
Les deux zones de texte s'actualisent à des intervalles différents.

Les tableaux

Nous avons vu que les contrôles ListBox exploitent les valeurs d'index. Chaque élément de la liste se voit attribuer une valeur d'index : `0` pour le premier, `1` pour le deuxième, et ainsi de suite. La zone de liste en elle-même est unique, mais elle peut recouvrir une multitude de valeurs. Ces valeurs se distinguent par leur valeur d'index.

Cette indexation des éléments de liste offrent une analogie intéressante avec le concept que nous allons maintenant étudier : les *tableaux*. La zone de liste est un contrôle avec des éléments indexés. Un tableau est une liste de variables qui contient des éléments indexés. La terminologie exacte nomme les valeurs d'index des tableaux des *indices*.

Un tableau est une liste de variables, de noms et de types de données identiques. Le terme indice désigne chaque variable de la liste.

L'indice est la valeur d'index d'un élément de tableau.

Les variables suivantes sont proprement individuelles et ne font pas partie d'un tableau :

```
intCount     curPay     sngLength     strDeptCode
```

Les variables individuelles se prêtent bien au stockage de valeurs individuelles (la prime accordée à tel ou tel commercial, etc.). En revanche, pour stocker une liste de valeurs similaires, on se servira plutôt d'un tableau.

Il ne s'agit pas, une fois que vous saurez déclarer et utiliser les tableaux, d'y stocker *toutes* vos données Visual Basic. Pour les boucles ou les entrées utilisateur, vous continuerez à employer les variables individuelles, faites pour cela. Les tableaux, eux, sont faits pour recevoir des listes de données. En cas d'occurrences multiples de données devant être suivies dans votre application (par exemple, les champs d'une table lue en mémoire), les tableaux permettront de stocker d'une façon adéquate les données.

Imaginons que vous ayez à établir des statistiques au sujet des primes accordées à 100 commerciaux. Vous devez calculer, pour toutes ces primes, la moyenne, le maximum, le minimum et la variation moyenne des valeurs. Vous pouvez déclarer 100 variables, chacune portant un nom différent, tel que `curBonus1`, `curBonus2`, etc. Et pour cumuler tout ça ? Il faudrait, pour chaque addition ou chaque comparaison, répéter les 100 variables dans une même instruction... C'est là qu'interviennent les tableaux. Au lieu de déclarer 100 variables, vous déclarez *un seul* tableau, portant un seul et même nom, et contenant la liste des 100 primes. Pour parcourir les valeurs du tableau, nul besoin de lister les 100 variables ; il suffit d'une boucle `For` qui passe d'un indice à l'autre, de `1` à `100`. Sans l'aide des tableaux, de telles manipulations seraient quasi impensables.

Pour mieux illustrer cet exemple, voici un fragment du code qui aurait servi à additionner les 100 variables individuelles :

```
curTotal = curBonus1 + curBonus2 + curBonus3 + curBonus4 + ...
```

Et voici, en comparaison, le code qui additionne les 100 valeurs du tableau :

```
For intCtr = 1 To 100
  curTotal = curTotal + curBonus(intCtr)
Next intCtr
```

La boucle aurait aussi bien pu parcourir les valeurs en sens inverse. En outre, il n'est pas nécessaire d'accéder aux éléments du tableau dans l'ordre. Supposons que vous vouliez calculer la moyenne de la première et de la dernière prime du tableau. C'est ce que fait l'instruction suivante :

```
curAvgBonus = (curBonus(1) + curBonus(100)) / 2.0
```

Cette simple ligne en dit déjà beaucoup sur les tableaux :

- Les indices sont indiqués entre parenthèses après le nom du tableau.
- Les éléments d'un tableau ont tous le même type de données.

- Les éléments du tableau peuvent être appelés dans n'importe quel ordre, pourvu qu'on spécifie l'indice.
- Les indices vont de `1` au nombre des éléments du tableau.

A dire vrai, ce dernier point dépend de vos préférences. Par défaut, le premier indice est `0`. Mais les programmeurs Visual Basic (contrairement aux programmeurs C et C++) préfèrent, en général, commencer par l'indice `1`. Pour ce faire, il suffit d'inclure dans la section de déclarations du module l'instruction suivante :

```
Option Base 1
```

Visual Basic offre un autre moyen de définir la plage des indices ; nous verrons cela dans la prochaine section.

Certains programmeurs Visual Basic appliquent aux tableaux une convention de dénomination plus précise ; ils emploient, par exemple, le préfixe `strar` pour les tableaux de chaînes (string array) et le préfixe `intar` pour les tableaux d'entiers (integer array). Libre à vous, naturellement, d'employer des préfixes francisés tels que `chtab` ou `entab`.

Visual Basic supporte deux types de tableaux : les tableaux statiques et les tableaux dynamiques. La taille des tableaux statiques est définitive et n'est pas modifable lors de l'exécution, alors que la taille des tableaux dynamiques est modifiable à tout moment. Nous ne traiterons, dans ce livre, que des tableaux statiques, beaucoup plus efficaces, et beaucoup plus utilisés.

D'une certaine manière, on accède aux valeurs d'un tableau comme le facteur distribue le courrier d'un immeuble : l'adresse est la même, seul change le nom des destinataires.

Déclaration des tableaux

Comme les variables individuelles, les tableaux doivent être déclarés avant de pouvoir être utilisés. On se sert, pour cela, des mêmes instructions de déclaration `Public` et `Dim`. Le choix de l'instruction dépend de la portée voulue et du lieu de déclaration du tableau.

`Public` permet de déclarer les tableaux publics, destinés à servir dans tous les modules de l'application. L'instruction `Public` doit apparaître dans la section de déclaration du module. Utilisée dans la section de déclaration du module, l'instruction `Dim` déclare un tableau de niveau module. Utilisée à l'intérieur d'une procédure, `Dim` déclare un tableau local.

Seule différence entre la déclaration des tableaux et la déclaration des variables : la présence dans l'instruction des indices. Voici le format des deux instructions :

```
Public arName(intSub) [As dataType][, arName(intSub)
[As dataType]]...
Dim arName(intSub) [As dataType][, arName(intSub) [As dataType]]...
```

On applique aux tableaux les mêmes conventions de dénomination qu'aux variables (à ceci près que vous pouvez préciser un deuxième préfixe). Les tableaux peuvent avoir n'importe quel type de données. Ici, *dataType* peut donc être `Integer`, `Single`, etc. *intSub* indique le nombre d'éléments, et la façon dont on se réfère à ces éléments. Dans les formats d'instructions présentés ci-dessus, *intSub* s'emploie de la manière suivante :

```
[intLow To] intHigh
```

(Où *intLow* représente la plage inférieure, et *intHigh* la plage supérieure.)

Nous ne parlons ici que des tableaux unidimensionnels, c'est-à-dire des tableaux à indice unique. Visual Basic supporte également les tableaux multi-dimensionnels, ou tables.

Il y a, en fait, (et c'est une particularité de Visual Basic) une exception à la règle selon laquelle un tableau doit relever d'un seul type de données. En effet, les tableaux peuvent être déclarés comme de type `Variant` ; auquel cas, les éléments contenus pourront être de types de données différents.

L'instruction suivante déclare un tableau de type `Integer` contenant cinq éléments (on présuppose ici que l'instruction `Option Base 1` apparaît dans la section de déclarations) :

```
Dim intCounts(5) As Integer
```

La Figure 10.9 illustre le stockage du tableau dans la mémoire.

Sans l'instruction `Option Base 1`, la déclaration `Dim` du tableau `intCounts` aurait donné un tableau à six éléments (indices `0` à `5`).

Qu'y a-t-il dans `intCounts(1)`, `intCounts(2)`, etc. ? Eh bien, personne ne le sait pour l'instant. Comme les variables, les éléments du tableau doivent avant toute chose être initialisés. En précisant l'indice, vous pouvez utiliser les éléments de tableau comme vous le feriez pour des variables. Par exemple :

```
intNumber = intCounts(2) * intFactor / 15
txtValue.Text = intCounts(4)
intCounts(5) = 0
```

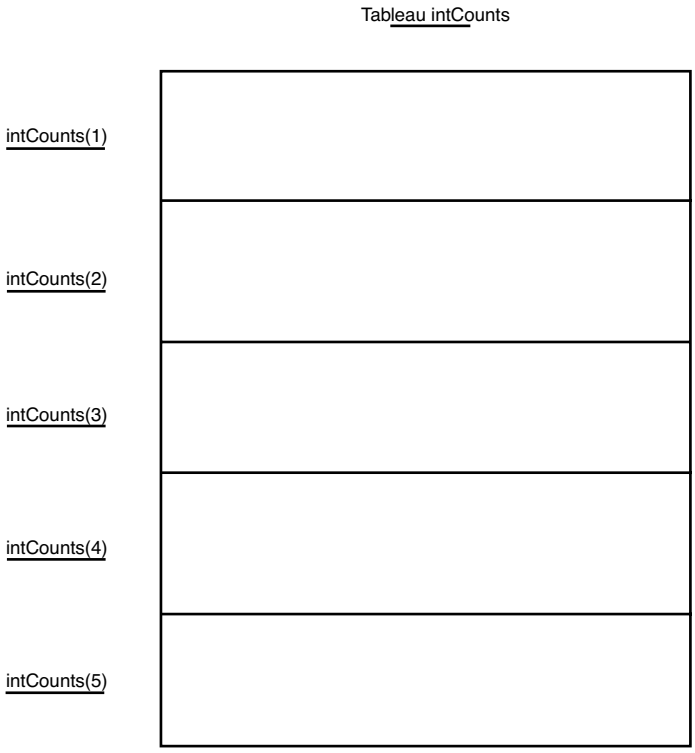


Figure 10.9
Le tableau `intCounts` contient cinq éléments ; le premier indice est `1`.

Les deux instructions suivantes sont équivalentes :

```
Dim intCounts(5) As Integer
Dim intCounts(1 To 5)
```

On peut aussi, à l'aide de la clause `To`, spécifier les indices de début et de fin. Considérez les trois déclarations de tableaux suivantes :

```
Public varCustNumber(200 To 999) As Variant
Public strCustName(200 To 999) As String
Public curCustBalance(200 To 999) As Currency
```

Dans chaque tableau, le premier indice est `200`, le dernier `999`. Si, après une telle déclaration, vous tentiez quelque chose comme `strCustName(4)`, Visual Basic générerait une erreur.

L'indice le plus haut spécifié par la clause `To` n'indique pas nécessairement le nombre d'éléments ; tout dépend de l'indice de départ. Nos trois tableaux, par exemple, contiennent chacun un total de 800 éléments.

Il pourra vous être utile, selon le cas, de spécifier un indice de départ autre que `0` ou `1`. La déclaration donnée en exemple ci-dessus, par exemple, serait appropriée à une liste de client dont le premier numéro de compte est 200. En faisant commencer vos indices à 200, vous assurez la symétrie entre les numéros de compte et les indices eux-mêmes.

Spécifiez toujours un indice de fin assez grand, afin de réserver une marge suffisante pour les éléments à venir.

Visual Basic inclut une fonction interne spéciale, `Array()`. La fonction `Array()` permet de déclarer et d'initialiser les tableaux plus facilement.

La fonction `Array()` est assez proche des anciennes instructions BASIC `READ` et `DATA`. Les petits tableaux dont les valeurs sont déjà connues peuvent être ainsi rapidement initialisés.

Le type de données `Variant`, vous le savez, peut contenir tous les autres types. Supposons que vous cherchiez à stocker dans un Tableau `Days` le nombre de jours de chaque mois (sans tenir compte des années bissextiles). Pour commencer, vous déclarez une variable `Variant` :

```
Dim Days As Variant
```

Puis, plutôt que d'utiliser une boucle `For`, vous initialisez le tableau en une seule étape, à l'aide de la fonction `Array()` :

```
Days = Array(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
```

Rappelons que, si l'instruction `Option Base 1` apparaît dans la section de déclarations du module, le premier indice de `Array()` est `1`. On peut déclarer et initialiser des chaînes et des dates de la même façon, en affectant à une valeur `Variant` la fonction `Array()`.

Exploitation des tableaux

Comme nous l'avons vu à la section précédente, les éléments de tableaux peuvent être impliqués dans des calculs, au même titre que les variables individuelles. Exemple :

```
curFamilyDues(5) = curFamilyDues(4) * 1.5
```

Pour traiter les données d'un tableau, il suffit de spécifier l'indice des éléments utilisés.

Pour vous familiariser avec l'utilisation des indices, nous vous proposons quelques exemples. Bien que la plupart des programmes reçoivent leurs données de fichiers et de formulaires, les listings qui suivent mettent en œuvre des tableaux dont les éléments sont des valeurs renvoyées par la fonction `InputBox()`. Les tableaux étant destinés à recevoir d'importantes quantités de données, il n'est pas question de saisir ces données dans les variables avant chaque exécution du programme. Les instructions d'affectation ne conviennent pas non plus aux grandes quantités de données exploitées par des programmes interactifs.

Dans le Listing 10.3, le programme déclare deux tableaux. Les données sont, d'une part, les noms de 35 familles membres d'une association de quartier, et, d'autre part, les cotisations annuelles dues. Le programme reçoit les données puis affiche les résultats.

Si vous devez exécuter ce programme, il vaut peut-être mieux réduire le nombre d'éléments de 35 à 5 afin de vous épargner une saisie inutile.

Listing 10.3 : Les tableaux simplifient le stockage des données

```
1:  Private Sub association ()
2:  ' Reçoit puis affiche les noms et les sommes dues.
3:    Dim strFamilyName(35) As String ' Réserve les éléments du tableau.
4:    Dim curFamilyDues(35) As Currency
5:    Dim intSub As Integer
6:    Dim intMsg As Integer ' Valeur de renvoi de MsgBox().
7:
8:    ' Collecte les données.
9:    For intSub = 1 To 35
10:     strFamilyName(intSub) = InputBox("Famille suivante ?")
21:     curFamilyDues(intSub) = InputBox("Cotisations dues ?")
22:   Next intSub
23:
24:   ' Les données peuvent maintenant être affichées.
25:   ' (Cet exemple utilise à cette fin des boîtes
26:   '  de message pour simplifier les choses.)
27:   intSub = 1 ' Initialise le premier indice.
28:   Do
29:     intMsg = MsgBox("Famille numéro  " & intSub & " : " &
        ➥strFamilyName(intSub))
30:     intMsg = MsgBox("Cotisations dues : " & curFamilyDues(intSub))
31:     intSub = intSub + 1
32:   Loop Until (intSub > 35)
33: End Sub
```

Notez que ce programme reçoit et affiche toutes les données à l'aide de simples routines. La routine d'entrée (lignes 9 à 22) utilise une boucle `For`, tandis que la routine de sortie (lignes 28 à 32) utilise une boucle `Do`. La méthode employée pour contrôler les

boucles est sans importance. Ce qui compte, ici, c'est que l'on reçoive, traite et affiche un grand nombre de données en un minimum de lignes, et que suffisent, pour cela, les indices du tableau et quelques instructions de boucles.

Cet exemple met en œuvre des *tableaux parallèles*, c'est-à-dire deux tableaux travaillant côte à côte. A chaque élément d'un tableau correspond un élément de l'autre tableau. Les tableaux parallèles fonctionnent en mémoire comme fonctionnent, dans une table, deux champs liés.

Notre petit programme d'association de quartier fournit une belle illustration, mais ne fonctionne que s'il y a exactement 35 familles. Et si l'association s'agrandit ? Il faudra alors modifier le programme. C'est pourquoi la plupart des programmes, contrairement à notre exemple, ne définissent pas a priori une quantité limite de données. Il vaut toujours mieux déclarer plus d'éléments de tableau que l'on n'a déjà de données. Le programme permet de déterminer quels éléments sont réellement utilisés.

Le prochain programme est semblable à celui du Listing 10.3, à ceci près qu'il déclare 500 éléments pour chaque tableau. Avec un tel nombre, on est certain d'avoir toujours assez de valeurs disponibles. L'utilisateur n'entre que le nombre d'élément réels. Le programme du Listing 10.4 est très flexible, permettant un nombre d'entrées et de sorties différent à chaque exécution. La seule limte est celle que l'on atteindra lorsque l'association approchera les 500 membres.

Déclarez assez d'éléments de tableaux pour couvrir vos besoins estimés, mais pas plus que vous ne puissiez réellement utiliser. Les éléments de tableaux occupent de la mémoire même s'ils sont vides.

Listing 10.4 : On peut déclarer plus d'éléments que l'on n'a de données

```
1:  Private Sub varyNumb ()
2:  ' Reçoit puis affiche les noms et les sommes dues.
3:    Dim strFamilyName(500) As String    ' On vise large.
4:    Dim curFamilyDues(500) As Currency
5:    Dim intSub As Integer, intNumFam As Integer
6:    Dim intMsg As Integer ' Valeur de renvoi de MsgBox().
7:    intNumFam = 1
8:    ' La boucle demande les noms et les sommes dues
9:    ' jusqu'à ce que l'utilisateur appuie sur Entrée
10:   ' sans avoir saisi d'information. Dès qu'une chaîne
11:   ' nulle est entrée, la boucle Do-Loop s'arrête
12:   ' après avoir stocké la dernière entrée.
13:   Do
14:     strFamilyName(intNumFam) = InputBox("Famille suivante ?")
15:     If (strFamilyName(intNumFam) = "") Then Exit Do      ' Interruption.
16:     curFamilyDues(intNumFam) = InputBox("Cotisations dues ?")
17:     intNumFam = intNumFam + 1 ' Ajoute 1 Add à la variable indice.
18:   Loop Until (intNumFam > 500)
```

Listing 10.4 : On peut déclarer plus d'éléments que l'on n'a de données *(suite)*

```
19:
20:  ' Lorsque la dernière boucle se termine, intNumFam contient
21:  ' 1 de plus que le nombre réel d'entrées.
22:
23:  ' Affiche toutes les données.
24:  For intSub = 1 To intNumFam - 1
25:     intMsg = MsgBox("Famille numéro  " & intSub & " : " &
        ➥strFamilyName(intSub))
26:     intMsg = MsgBox("Cotisations dues : " & curFamilyDues(intSub))
27: Next intSub
28: End Sub
```

A la ligne 15, la frappe de la touche Entrée sans saisie de valeur provoque l'interruption de la boucle. Ce n'est pas parce que 500 éléments sont réservés qu'il faut tous les utiliser.

Le programme du Listing 10.5 montre comment on accède en désordre aux éléments du tableau. Le programme demande les salaires payés pour chacun des douze derniers mois. Il attend ensuite que l'utilisateur saisisse le mois à consulter. Le total du mois demandé s'affiche alors. Ce listing donne un exemple de programme capable de rechercher des données précises dans un tableau : on stocke les données, puis on attend la requête de l'utilisateur.

Listing 10.5 : Programme de recherche de données

```
1:  Private Sub salary ()
2:  ' Stocke 12 mois de salaires, puis affiche les mois sélectionnés.
3:    Dim curSal(1 To 12) As Currency   ' Réserve des éléments
      ➥pour 12 salaires.
4:    Dim intSub As Integer   ' Indice de boucle.
5:    Dim intNum As Integer   ' Mois sélectionné.
6:    Dim intMsg As Integer   ' Valeur de renvoi de MsgBox().
7:    Dim strAns As String
8:
9:    For intSub = 1 To 12
10:     curSal(intSub) = InputBox("Salaire pour le mois de " &
        ➥Str(intSub) & " ?", 0.00)
11:   Next intSub
12:
13:   ' Demande le numero du mois.
14:   Do
15:     intNum = InputBox("Quel mois voulez-vous consulter ? (1-12) ")
16:     intMsg = MsgBox("Salaires pour le mois de " & Str(intNum)
        ➥& " : " & curSal(intNum))
17:   strAns = InputBox("Autre consultation ? (O/N)")
18:   Loop While (strAns = "O" Or strAns = "o")
19: End Sub
```

Après que l'utilisateur a entré les douze salaires dans le tableau (lignes 9 à 11), il peut demander à consulter n'importe quel mois (un à la fois), simplement en spécifiant le numéro du mois, qui correspond à l'indice.

Le Listing 10.6 illustre les diverses opérations mathématiques que l'on peut effectuer sur les tableaux. Le programme demande une série de températures, et interrompt sa demande lorsque l'utilisateur entre `-99` pour signaler la fin de la liste. Puis le programme calcule la moyenne des températures.

Listing 10.6 : L'utilisateur indique au programme la fin de la série de données

```
1:  Private Sub tempAvg ()
2:  ' Demande une liste de températures puis calcule la moyenne.
3:    Dim sngTemp(1 To 50) As Single ' Maximum = 50
4:    Dim sngTotalTemp As Single    ' Reçoit le total.
5:    Dim sngAvgTemp As Single
6:    Dim intSub As Integer         ' Indice.
7:    Dim intMsg As Integer         ' Valeur de renvoi de MsgBox().
8:
9:    ' Demande à l'utilisateur chaque température.
10:   For intSub = 1 To 50   ' Maximum.
11:      sngTemp(intSub) = InputBox("Température suivante ?
         (-99 pour terminer) ")
12:      ' Si l'utilisateur veut arrêter, décrémente de 1 et
         ➥sort de la boucle.
13:      If (sngTemp(intSub) = -99) Then
14:        intSub = intSub - 1   ' Décrémente de 1.
15:        Exit For
16:      End If
17:      sngTotalTemp = sngTotalTemp + sngTemp(intSub)
         ➥' Additionne le total.
18:
19:   Next intSub
20:   ' Calcule la moyenne.
21:   sngAvgTemp = sngTotalTemp / intSub
22:   intMsg = MsgBox("Température moyenne : " & sngAvgTemp)
23: End Sub
```

Les tableaux de contrôles

Vous rencontrerez, dans la suite de cet ouvrage, le terme *tableau de contrôles*. Un tableau de contrôles n'est rien d'autre qu'une liste de contrôles, tout comme un tableau de variables est une liste de variables. L'intérêt des tableaux de contrôles est le même que celui des tableaux de variables : on parcourt les données à l'aide d'une boucle plutôt que de saisir le nom de chaque contrôle individuel.

Nous n'en dirons pas plus pour l'instant. Les chapitres ultérieurs, tels que le Chapitre 16, reviendront de façon plus approfondie sur ce concept.

En résumé

Ce chapitre vous a expliqué comment intégrer le support de la souris à vos applications. Les événements souris informent votre programme des déplacements, clics et double-clics, et indiquent quel bouton a été utilisé. A l'aide des méthodes appropriées, vous pouvez gérer les opérations de glisser-déposer, qui permettent à l'utilisateur de déplacer un objet tel qu'un contrôle, d'une partie à l'autre de la feuille.

Nous avons également découvert le contrôle Timer, qui permet de programmer l'exécution selon des critères temporels. Le timer se base sur l'horloge interne du PC, et peut déclencher des procédures après écoulement d'un laps de temps défini en millisecondes.

Il existe plusieurs types de contrôles de listes La Boîte à outils comprend à cet effet les contrôles ListBox et ComboBox. La différence entre les types disponibles est parfois ténue, mais les exemples d'aujourd'hui auront détaillé le fonctionnement et les avantages de chacun, afin que vous fassiez le meilleur choix pour vos applications.

Vers la fin de ce chapitre, nous sommes passé des contrôles de listes aux tableaux de variables, dont la structure est semblable à celle des éléments de listes. Un tableau de variables contient de multiples valeurs, auxquelles on accède à l'aide d'un indice numérique.

Dans le prochain chapitre, nous étudierons la gestion des feuilles.

Questions-réponses

Q Comment choisir entre les différents types de listes ?

R Si les contrôles de listes s'obtiennent tous à partir des outils ListBox et ComboBox de la Boîte à outils, ils ont chacun un fonctionnement particulier. Dans la zone de liste simple, l'utilisateur ne peut que sélectionner l'un des éléments proposés, il ne

peut pas en ajouter. La liste conserve la taille qui lui a été attribuée lors de la création ; si nécessaire, des barres de défilement s'afficheront automatiquement.

Le contrôle ComboBox, selon les spécifications de sa propriété `Style`, peut revêtir trois formes différentes. La ComboBox déroulante reste fermée jusqu'à ce que l'utilisateur clique dessus, ce qui économise de la place sur la feuille. La ComboBox simple inclut une zone de texte dans laquelle l'utilisateur peut entrer de nouvelles valeurs. De même, la zone de liste déroulante n'occupe que peu de place, et autorise l'ajout d'éléments.

Q Pour présenter à l'utilisateur une liste de valeurs, dois-je utiliser l'une des zones de liste ou un tableau ?

R Ce sont deux choses différentes, et vous n'avez pas à choisir entre zones de liste et tableaux. Vous pouvez très bien utiliser les deux dans une même application. Le tableau est là pour accueillir les données reçues par votre application, tandis que les listes ne servent qu'à présenter des données.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quels événements souris répondent à des boutons particuliers ? Quels événements souris répondent aux deux ?
2. Comment déterminer quel bouton a servi lors d'un événement `MouseDown` ?
3. Comment modifier l'icône qui apparaît lors d'une opération de glisser-déposer ?
4. Comment créer une procédure événementielle capable de répondre à un intervalle plus grand que 65 secondes (limite approximative du contrôle Timer) ?
5. Comment initialise-t-on un contrôle de liste ?
6. Comment le programme reconnaît-il les éléments que l'utilisateur a sélectionnés ?
7. Donnez deux moyens de retirer tous les éléments d'un contrôle de liste.

8. Quand un contrôle ComboBox permet à l'utilisateur d'entrer de nouvelles valeurs, pourquoi faut-il fournir au moins un autre contrôle ?
9. Comment faire en sorte qu'un contrôle de liste affiche les éléments dans l'ordre alphabétique ou numérique, indépendamment de l'ordre dans lequel on les a entrés ?
10. Combien d'éléments sont réservés par l'instruction `Dim` suivante ?

```
Dim varStaff(-18 To 4) As Variant
```

Exercices

1. Ecrivez un programme qui affiche dans une zone de liste la liste des membres de votre famille. Ajoutez-en assez pour que des barres de défilement s'affichent. Faites en sorte que la liste soit automatiquement triée.
2. Lancez le projet d'exemple `Listcmbo`, fourni avec Visual Basic. L'application contient une base de données d'éditeurs informatiques américains. Cliquez sur le bouton Utiliser le contrôle ComboBox standard, puis cliquez sur le champ Etat. Cliquez ensuite sur le bouton Utiliser le contrôle ListBox standard, puis ouvrez de nouveau le champ Etat. Vous constaterez aisément la différence entre les deux.
3. Modifiez l'application de l'exercice 1 de façon que les trois sortes de ComboBox apparaissent sur la feuille, pour afficher les mêmes données. Faites en sorte que tout changement (ajout d'élément) dans l'une des ComboBox soit automatiquement reporté dans l'autre.

PB4

Sélections multiples dans une zone de liste

Ce Projet bonus met en œuvre les zones de liste étudiées au Chapitre 10. Il vous permettra également d'approfondir votre compréhension des tableaux. L'application que vous allez créer doit permettre les sélections multiples ; vous devrez pour cela définir les valeurs de propriétés adéquates. Vous devrez également déterminer si l'utilisateur a bel et bien sélectionné plusieurs valeurs, afin que ces valeurs puissent servir ailleurs dans le programme.

Les sélections multiples sont souvent utilisées. Par exemple, une entreprise peut distribuer à ses clients un fichier catalogue dans lequel les divers produits sont proposés sous forme de liste. Le client peut décider de n'acheter qu'un seul produit, comme en sélectionner des dizaines.

Créer la feuille

Pour qu'une zone de liste accepte les sélections multiples, vous devez modifier la propriété `MultiSelect`. Si la `MultiSelect` a la valeur `1 - Simple` (comme opposée de la valeur par défaut, `0 - None`), l'utilisateur peut effectuer des sélections multiples. Si la valeur de `MultiSelect` est `2 - Extended`, l'utilisateur peut sélectionner une plage d'éléments en tenant la touche Maj, ainsi que sélectionner plusieurs éléments discontinus en tenant la touche Ctrl (exactement comme dans les boîtes de dialogue Ouvrir).

La Figure PB4.1 montre une feuille sur laquelle des zones de texte affichent "Sélectionné" si l'élément correspondant a été sélectionné dans la zone de liste, et "Non sélectionné" autrement. La propriété `MultiSelect` de la liste "Destination" a pour valeur `2 - Extended`.

Figure PB4.1
Vous pouvez effectuer des sélections multiples dans les zones de liste.

La première chose à faire, c'est d'ajouter les contrôles sur la feuille. Suivez pour cela le Tableau PB4.1.

Tableau PB4.1 : Propriétés des contrôles de la feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmDest`
Feuille	`Caption`	`Démo zones de liste`
Feuille	`Height`	`6600`
Feuille	`Width`	`7230`
Label 1	`Name`	`lblListBoxCap`
Label 1	`Caption`	`Destination`
Label 1	`FontStyle`	`Bold`
Label 1	`FontSize`	`24`

Tableau PB4.1 : Propriétés des contrôles de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Label 1	`Height`	`600`
Label 1	`Left`	`2190`
Label 1	`Top`	`120`
Label 1	`Width`	`3060`
Label 2	`Name`	`lblChicago`
Label 2	`BorderStyle`	`Fixed Single`
Label 2	`Caption`	`Chicago`
Label 2	`FontStyle`	`Bold`
Label 2	`FontSize`	`18`
Label 2	`Height`	`495`
Label 2	`Left`	`360`
Label 2	`Top`	`1995`
Label 2	`Width`	`2655`
Label 3	`Name`	`lblWashington`
Label 3	`Caption`	`Washington`
Label 3	`BorderStyle`	`Fixed Single`
Label 3	`FontStyle`	`Bold`
Label 3	`FontSize`	`18`
Label 3	`Height`	`495`
Label 3	`Left`	`4035`
Label 3	`Top`	`1980`
Label 3	`Width`	`2655`
Label 4	`Name`	`lblDallas`

Tableau PB4.1 : Propriétés des contrôles de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Label 4	`Caption`	`Dallas`
Label 4	`BorderStyle`	`Fixed Single`
Label 4	`FontStyle`	`Bold`
Label 4	`FontSize`	`18`
Label 4	`Height`	`495`
Label 4	`Left`	`360`
Label 4	`Top`	`3480`
Label 4	`Width`	`2655`
Label 5	`Name`	`lblHouston`
Label 5	`Caption`	`Houston`
Label 5	`BorderStyle`	`Fixed Single`
Label 5	`FontStyle`	`Bold`
Label 5	`FontSize`	`18`
Label 5	`Height`	`495`
Label 5	`Left`	`4035`
Label 5	`Top`	`3480`
Label 5	`Width`	`2655`
Label 6	`Name`	`lblSeattle`
Label 6	`Caption`	`Seattle`
Label 6	`BorderStyle`	`Fixed Single`
Label 6	`FontStyle`	`Bold`
Label 6	`FontSize`	`18`
Label 6	`Height`	`495`

Tableau PB4.1 : Propriétés des contrôles de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Label 6	`Left`	`360`
Label 6	`Top`	`4920`
Label 6	`Width`	`2655`
Label 7	`Name`	`lblDayton`
Label 7	`Caption`	`Dayton`
Label 7	`BorderStyle`	`Fixed Single`
Label 7	`FontStyle`	`Bold`
Label 7	`FontSize`	`18`
Label 7	`Height`	`495`
Label 7	`Left`	`4035`
Label 7	`Top`	`4920`
Label 7	`Width`	`2655`
Zone de liste	`Name`	`lstFirstList`
Zone de liste	`Height`	`840`
Zone de liste	`Left`	`2865`
Zone de liste	`MultiSelect`	`2-Extended`
Zone de liste	`Top`	`870`
Zone de liste	`Width`	`1335`
Zone de texte 1	`Name`	`txtChicago`
Zone de texte 1	`FontSize`	`18`
Zone de texte 1	`FontStyle`	`Bold`
Zone de texte 1	`Height`	`495`
Zone de texte 1	`Left`	`120`

Tableau PB4.1 : Propriétés des contrôles de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Zone de texte 1	`Text`	`Non sélectionné`
Zone de texte 1	`Top`	`2520`
Zone de texte 1	`Width`	`3105`
Zone de texte 2	`Name`	`txtWashington`
Zone de texte 2	`FontSize`	`18`
Zone de texte 2	`FontStyle`	`Bold`
Zone de texte 2	`Height`	`495`
Zone de texte 2	`Left`	`3840`
Zone de texte 2	`Text`	`Non sélectionné`
Zone de texte 2	`Top`	`2520`
Zone de texte 2	`Width`	`3105`
Zone de texte 3	`Name`	`txtDallas`
Zone de texte 3	`FontSize`	`18`
Zone de texte 3	`FontStyle`	`Bold`
Zone de texte 3	`Height`	`495`
Zone de texte 3	`Left`	`120`
Zone de texte 3	`Text`	`Non sélectionné`
Zone de texte 3	`Top`	`3960`
Zone de texte 3	`Width`	`3105`
Zone de texte 4	`Name`	`txtHouston`
Zone de texte 4	`FontSize`	`18`
Zone de texte 4	`FontStyle`	`Bold`
Zone de texte 4	`Height`	`495`

Tableau PB4.1 : Propriétés des contrôles de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Zone de texte 4	`Left`	`3840`
Zone de texte 4	`Text`	`Non sélectionné`
Zone de texte 4	`Top`	`3960`
Zone de texte 4	`Width`	`3105`
Zone de texte 5	`Name`	`txtSeattle`
Zone de texte 5	`FontSize`	`18`
Zone de texte 5	`FontStyle`	`Bold`
Zone de texte 5	`Left`	`120`
Zone de texte 5	`Height`	`495`
Zone de texte 5	`Text`	`Non sélectionné`
Zone de texte 5	`Top`	`5400`
Zone de texte 5	`Width`	`3105`
Zone de texte 6	`Name`	`txtDayton`
Zone de texte 6	`FontSize`	`18`
Zone de texte 6	`FontStyle`	`Bold`
Zone de texte 6	`Height`	`495`
Zone de texte 6	`Left`	`3840`
Zone de texte 6	`Text`	`Non sélectionné`
Zone de texte 6	`Top`	`5400`
Zone de texte 6	`Width`	`3105`

Ajouter le code

Le Tableau PB4.1 ne réserve pas de blancs dans les valeurs des zones de texte ; ce sera le rôle de la procédure `Form_Load()`. Le Listing PB4.1 fournit le code de ce projet.

Listing PB4.1 : Initialiser la zone de liste et interroger les sélections multiples

```
1:  Private Sub Form_Load()
2:    ' S'exécute au chargement de la feuille.
3:    lstFirstList.AddItem "Chicago"
4:    lstFirstList.AddItem "Dallas"
5:    lstFirstList.AddItem "Seattle"
6:    lstFirstList.AddItem "Washington"
7:    lstFirstList.AddItem "Houston"
8:    lstFirstList.AddItem "Dayton"
9:  End Sub
10:
11: Private Sub lstFirstList_Click()
12:   ' Met à jour les six zones de texte en fonction des
13:   ' éléments sélectionnés dans la première zone de liste.
14:   If lstFirstList.Selected(0) Then
15:     txtChicago.Text = "Sélectionné"
16:   Else
17:     txtChicago.Text = "Non sélectionné"
18:   End If
19:
20:  If lstFirstList.Selected(1) Then
21:     txtDallas.Text = "Sélectionné"
22:   Else
23:     txtDallas.Text = "Non sélectionné"
24:   End If
25:
26:   If lstFirstList.Selected(2) Then
27:     txtSeattle.Text = "Sélectionné"
28:   Else
29:     txtSeattle.Text = "Non sélectionné"
30:  End If
31:
32:   If lstFirstList.Selected(3) Then
33:     txtWashington.Text = "Sélectionné"
34:   Else
35:     txtWashington.Text = "Non sélectionné"
36:   End If
37:
38:   If lstFirstList.Selected(4) Then
39:     txtHouston.Text = "Sélectionné"
40:  Else
41:     txtHouston.Text = "Non sélectionné"
42:   End If
43:
```

```
44:   If lstFirstList.Selected(5) Then
45:     txtDayton.Text = "Sélectionné"
46:   Else
47:     txtDayton.Text = "Non sélectionné"
48:   End If
49:
50: End Sub
```

Analyse

Les lignes 3 à 8 initialisent la zone de liste. La propriété `MultiSelect` ayant pour valeur `Extended`, l'utilisateur peut sélectionner plusieurs villes à la fois.

Lorsque les sélections multiples sont autorisées, Visual Basic doit créer un tableau spécial, semblable aux tableaux de variables, mais ne contenant que des propriétés. Ici, le tableau de propriétés est nommé `Selected`. Ce tableau s'étend de `Selected(0)` à `Selected(5)`, pour accueillir les six éléments de la liste.

Les valeurs du tableau sont toutes de type `Boolean`. Elles seront `True` ou `False`, selon la sélection. A la première exécution du programme, toutes les valeurs `Selected` sont `False`, puisque aucune sélection n'a encore eu lieu. Chaque fois que l'utilisateur sélectionne un élément dans la liste, la valeur `Selected` correspondante devient `True`. La procédure `Click()` (lignes 11 à 48) se charge de mettre à jour les six zones de texte pour refléter les sélections.

Si l'utilisateur déséléctionne un élément en y cliquant de nouveau, la valeur `Selected` correspondante redevient `False`.

PB 5

Pratique de la souris

Ce Projet bonus vous invite à mettre en pratique les techniques de gestion de la souris étudiées au Chapitre 10. En répondant aux événements souris et en interrogeant les arguments de la procédure événementielle, le programme détermine si l'utilisateur a cliqué, double-cliqué, ou s'il a déplacé la souris.

Ce Projet bonus est un peu différent des autres. Vous commencerez par étudier un exemple dans lequel on change l'icône de déplacement. Puis, vous construirez une application capable de répondre aux événements souris.

Changer l'icône pointeur

Comme nous l'avons vu, le pointeur peut changer d'apparence lorsqu'il survole un contrôle. Nous allons créer, à titre d'exemple, une application simplissime : une feuille unique avec un bouton de commande au milieu. Il s'agit de faire en sorte que le pointeur se transforme en "smiley" lorsqu'il est positionné sur le bouton de commande `cmdHappy`. Dans la fenêtre Propriétés, affectez au bouton une image du dossier \Graphics\Icons\Misc. Le fichier Face03.ico sera parfait.

Maintenant, définissez la propriété `MousePointer` du bouton de commande comme `99 - Custom`. Cette valeur ordonne à Visual Basic d'afficher l'icône spécifiée dans la propriété `MouseIcon` *si et seulement si* l'utilisateur place le pointeur sur le bouton de commande. La Figure PB5.1 montre le pointeur "smiley".

Figure PB5.1
Voilà un pointeur heu-reux !

Lancez cette petite application, et vous constaterez que Visual Basic se charge d'afficher l'icône spécifiée.

Programmer la souris

Nous allons maintenant créer un nouveau projet dans lequel un contrôle affiche une image (comme cela a été étudié au Chapitre 2). A la propriété `Picture` de ce contrôle Image, vous affecterez une icône figurant une cible. Cette icône se trouve dans le dossier \Graphics\Icons\Misc. Vous ajouterez ensuite à la feuille une zone de texte nommée `txtMouse` (voir Tableau PB5.1). Le contenu de cette zone de texte reflétera les déplacements et les clics de la souris. Ce projet illustre le fonctionnement des procédures événementielles liées à la souris.

La Figure PB5.2 montre la feuille que vous allez créer.

Figure PB5.2
L'utilisateur doit cliquer sur l'image.

Il faut d'abord créer la feuille. Suivez, pour cela, les spécifications du Tableau PB5.1.

Tableau PB5.1 : Propriétés des contrôles de la feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmMouse`
Feuille	`Caption`	`Gestion de la souris`
Feuille	`Height`	`4230`
Feuille	`Width`	`5265`
Zone de texte	`Name`	`txtMouse`
Zone de texte	`Alignment`	`2-Center`
Zone de texte	`FontStyle`	`Bold`
Zone de texte	`FontSize`	`14`
Zone de texte	`Height`	`1095`
Zone de texte	`Left`	`840`
Zone de texte	`MultiLine`	`True`
Zone de texte	`Text`	`Servez-vous de la souris`
Zone de texte	`Top`	`1320`
Zone de texte	`Width`	`3255`
Image	`Name`	`imgMouse`
Image	`Height`	`480`
Image	`Left`	`2400`
Image	`Picture`	`\Program Files\Microsoft VisualStudio\Common \Graphics\Icons\Misc\Bullseye`
Image	`Top`	`480`

Ajouter le code

Le Listing PB5.1 fournit le code de ce projet.

Listing PB5.1 : Répondre aux événements souris

```
1:  Private Sub Form_Click()
2:    txtMouse.Text = "Vous avez cliqué sur la feuille"
3:    Beep    ' Signale l'événement Click.
4:  End Sub
5:  Private Sub Form_DblClick()
6:    txtMouse.Text = "Vous avez double-cliqué sur la feuille"
7:  End Sub
8:
9:  Private Sub Form_MouseDown(intButton As Integer, intShift
    ➥As Integer, sngX As Single, sngY As Single)
10:   ' Clic sur la feuille.
11:    txtMouse.Text = "Clic sur la feuille à la position "
       ➥& sngX & "," & sngY
12: End Sub
13: ' Arguments ignorés dans la procédure précédente.
14:
15:  Private Sub Form_MouseMove(intButton As Integer, intShift
     ➥As Integer, sngX As  Single, sngY As Single)
16:    txtMouse.Text = "Déplacement de la souris..."
17:  End Sub
18:  ' Arguments ignorés dans la procédure précédente.
19:
20: Private Sub imgMouse_Click()
21:   txtMouse.Text = "Vous avez cliqué sur l'image"
22: End Sub
23:
24: Private Sub imgMouse_DblClick()
25:   txtMouse.Text = "Vous avez double-cliqué sur l'image"
26: End Sub
27:
28: Private Sub imgMouse_MouseDown(intButton As Integer, intShift
    ➥As Integer, sngX As Single, sngY As Single)
29: '
30: ' Clic sur l'image.
31:   txtMouse.Text = "Clic sur l'image à la position " & sngX & "," & sngY
32: End Sub
33:
34: Private Sub imgMouse_MouseMove(intButton As Integer, intShift
    ➥As Integer, sngX As Single, sngY As Single)
35:   txtMouse.Text = "Vous vous êtes déplacé sur l'image"
36: End Sub
```

Analyse

Les différentes procédures événementielles interrogent l'activité de la souris. Les lignes 1 à 4 contiennent la procédure événementielle la plus simple, qui répond lorsque que l'on clique n'importe où sur la feuille. L'instruction `Beep` est là pour signaler l'événement `Click`, qui autrement ne serait jamais perçu (il se produit bien trop rapidement). Comme nous l'avons vu au Chapitre 10, l'événement `MouseDown` a lieu avant l'événement `Click` (si, du moins, le code contient les deux événements). Le texte de `MouseDown` s'affichera donc pour montrer où l'on a cliqué sur la feuille. Puis, lorsqu'on relâche le bouton de la souris, c'est le texte de l'événement `Click` qui s'affiche pour disparaître aussitôt (d'où l'intérêt de l'instruction `Beep`).

Il est intéressant de noter qu'un double-clic sur la feuille déclenche d'abord l'événement `MouseDown`, puis l'événement `Click`, puis seulement l'événement `DblClick` (lignes 9 à 12). Dès que vous double-cliquez, le `Beep` se fait entendre. Mais vous verrez, avant, les coordonnées s'afficher sur ordre de l'événement `MouseDown`.

Il y a dans tout cela un défaut qui irrite beaucoup de programmeurs : l'événement `MouseMove` se déclenche chaque fois que l'on clique.

Les lignes 9 à 12 traitent les arguments de l'événement `MouseDown`. Les valeurs en twips `sngX` et `sngY` indiquent les coordonnées du clic. Notez que ce code réagira aux clics du bouton gauche comme aux clics du bouton droit, les deux pouvant déclencher l'événement `MouseDown`. La procédure ne s'exécute que si vous cliquez sur la feuille ; et l'événement `Click` a lieu dès que vous relâchez le bouton.

Les lignes 15 à 17 gèrent le déplacement de la souris sur la feuille, et uniquement sur la feuille.

La ligne 20 à 22 gèrent le positionnement de la souris sur l'image. Si vous cliquez sur l'image, la procédure événementielle `Click` met à jour la zone de texte pour indiquer ce qui vient de se passer. Mais l'événement `MouseDown` passant avant l'événement `Click`, vous n'aurez pas le temps de lire cette indication ; en revanche, vous pourrez entendre le son généré par l'instruction `Beep`.

Les lignes 24 à 26 gèrent le double-clic sur l'image à l'aide d'une procédure événementielle `DblClick`. Les lignes 28 à 32 traitent l'événement `MouseDown` de l'image, avant même que l'événement `Click` n'ait lieu. Enfin, les lignes 34 à 36 gèrent les mouvements de la souris sur l'image.

Nous aurions pu afficher dans la zone de texte les coordonnées de l'événement `MouseMove`*, mais ces informations auraient changé trop vite pour qu'on puisse les lire.*

Grâce à ce Projet bonus, vous savez à quoi vous attendre quant à la programmation de la souris. Vous savez notamment que l'événement `MouseDown` a toujours priorité sur les événements `Click` et `DblClick`.

Implémenter le glisser-déposer automatique

Rien de plus simple que d'ajouter à vos applications le support du glisser-déposer. D'abord, affectez à la propriété `DragIcon` du contrôle image l'icône qui doit s'afficher lors de l'opération. Choisissez, par exemple, l'image Clock02.ico. Pour implémenter le glisser-déposer automatique, définissez la propriété `DragMode` du contrôle comme `1 - Automatic`. Visual Basic s'occupe des détails.

Maintenant, ajoutez la procédure événementielle `MouseDown` suivante :

```
Private Sub Form_DragDrop(cntSource As Control, sngX As Single, sngY As Single)
cntSource.Move sngX, sngY  ' Déposer l'image.
End Sub
```

L'argument `Source` de l'événement `DragDrop` correspond en fait au contrôle glisser-déposer. Les valeurs de coordonnées `sngX` et `sngY` spécifient le "point de chute" du contrôle. La méthode `Move` se charge de déplacer le contrôle (en l'occurrence, l'image) à la position spécifiée.

Lancez l'application et déplacez l'image à divers endroits de la feuille. Le mode automatique requiert de votre part un minimum de programmation, et sera donc privilégié pour la plupart des opérations de glisser-déposer.

Implémenter le glisser-déposer manuel

Pour mettre en œuvre le glisser-déposer manuel, vous devez définir la propriété `DragMode` comme `0 - Manual`, puis remplacer le contenu de la procédure événementielle `MouseDown` et faire apparaître le code suivant :

```
Private Sub imgMouse_MouseDown(intButton As Integer, intShift
➥As Integer, sngX As Single, sngY As Single)
```

```
    ' Clic sur l'image.
    txtMouse.Text = "Clic sur l'image à la position " & sngX & "," & sngY
    imgMouse.Drag     ' Amorce le glisser-déposer.
End Sub
```

La seule instruction nouvelle ici est `imgMouse.Drag`. L'événement `MouseDown` doit amorcer l'opération de glisser-déposer. L'intérêt du mode manuel est que vous pouvez programmer divers traitements en réponse à l'événement `MouseDown`, avant que ne commence le glisser-déposer. En mode automatique, le glisser-déposer a lieu sans autre intervention de votre part.

Utilisez le glisser-déposer manuel lorsque vous souhaitez effectuer diverses tâches conjointement au glisser-déposer lui-même.

La méthode `Drag` active le glisser-déposer. Sans la méthode `Drag`, la procédure événementielle `MouseDown()` ne pourrait amorcer l'opération. Le mode manuel permet d'appliquer des restrictions à l'opération de glisser-déposer avant ou pendant qu'elle a lieu.

Chapitre 11

Gestion des feuilles

Dans ce chapitre, nous étudierons les feuilles de façon plus approfondie qu'il n'était possible jusque-là. Vous apprendrez à programmer et à gérer les feuilles à l'aide des propriétés, des événements et des méthodes appropriés. L'une des méthodes les plus utiles est `Print`, qui permet d'afficher du texte sur la feuille directement, sans passer par un contrôle. Il est intéressant de noter que la méthode `Print` vient tout droit du bon vieux langage BASIC, dont la commande `PRINT` fonctionnait exactement de la même manière.

Nous examinerons également les avantages et désavantages respectifs des applications MDI (monodocuments) et SDI (multidocuments). Enfin, vous apprendrez à placer sur vos feuilles des barres d'outils et des contrôles Coolbar, pour offrir à l'utilisateur un moyen supplémentaire d'interagir avec le programme.

Voici ce que nous découvrirons aujourd'hui :

- les propriétés, événements et méthodes des feuilles ;
- les collections de feuilles ;
- l'affichage de texte sur la feuille à l'aide de la méthode `Print` ;
- les différences entre les applications MDI et SDI ;
- comment créer vos propres propriétés de feuilles ;
- le contrôle Toolbar ;
- le contrôle Coolbar ;
- le contrôle ImagCombo.

Propriétés, événements et méthodes

Au cours des dix précédents chapitres, vous avez découvert une multitude de contrôles et de commandes. Vous êtes donc en passe de devenir un vrai pro de la programmation Visual Basic. Mais il vous reste à maîtriser l'élément le plus important d'une application Visual Basic : les feuilles. Les feuilles que nous n'avons fait qu'évoquer çà et là pour les besoins de notre exposé, méritent bien un chapitre entier.

Vous avez déjà appris à régler certaines propriétés pour modifier l'aspect et le comportement d'une feuille. Vous savez que l'on peut y ajouter un libellé, en définir la taille et spécifier si oui ou non des boutons de contrôles (tels qu'Agrandir et Réduire) seront disponibles. Vous avez également utilisé la procédure événementielle `Form_Load()` pour initialiser les contrôles de listes. C'est en effet dans cette procédure que l'on procède aux initialisations préalables à l'affichage de la feuille. (Le nom de la feuille n'est pas nécessaire, car le code apparaît dans le module de feuille et s'applique donc toujours à la feuille courante.)

La procédure événementielle `Form_Unload()` permet de faire "place nette" dans l'écran et la mémoire de l'utilisateur, lorsqu'une feuille se referme. A la différence des méthodes `Form.Hide` et `Form.Show`, les procédures événementielles `Form_Load()` et `Form_Unload()` interviennnent sur la mémoire. Lors de l'exécution, `Form_Load()` charge la feuille en mémoire, et `Form_Unload()` fait le ménage. Les méthodes `Show` et `Hide` ne sont que des avatars des valeurs `True` et `False` de la propriété `Visible` d'une feuille.

Faire
Utilisez les méthodes `Form.Hide` et `Form.Show` pour masquer et afficher les feuilles. N'utilisez `Form_Load()` et `Form_Unload()` que pour charger et décharger les feuilles de la mémoire.

Vous découvrirez, dans la prochaine section, une façon particulière d'employer l'événement `Form_Unload()`.

On rencontre souvent, dans une procédure `Form_Load()`, un code semblable à celui-ci :

```
frmAForm.Left = (Screen.Width - frmAForm.Width) / 2
frmAForm.Top = (Screen.Height - frmAForm.Height) / 2
```

Ces instructions permettent de centrer la feuille par rapport aux coordonnées de l'écran. Le Chapitre 16 vous présentera les divers types d'objets supportés par Visual Basic. L'un de ces objets est `Screen`, qui représente l'écran de l'utilisateur. L'objet `Screen` s'ajuste à la résolution d'écran et à la carte vidéo de l'utilisateur. Pour connaître la

résolution employée, le programme peut, à tout moment, interroger les propriétés `Width` et `Height` de l'objet `Screen`.

Vous vous épargnerez de la saisie en omettant l'objet par défaut dans la procédure événementielle. Ainsi, plutôt que de taper le nom de la feuille, vous pourriez réécrire les instructions ci-dessus de la manière suivante :

```
Left = (Screen.Width - Width) / 2
Top = (Screen.Height - Height) / 2
```

Dans la mesure où ce code apparaît dans une procédure événementielle de la feuille, telle que `Form_Load()`, vous n'avez pas à spécifier le nom de la feuille. En revanche, les économies de saisie se payent presque toujours par une certaine ambiguïté du code. En spécifiant tous les objets, même les objets par défaut, vous assurez la clarté du code et facilitez la maintenance. Que de cruels dilemmes en Visual Basic !

L'objet par défaut est l'objet (feuille, contrôle, etc.) que Visual Basic utilisera automatiquement si vous n'en spécifiez pas d'autre.

Le code de centrage présenté ci-haut forme un objet d'étude intéressant, car il illustre les relations qu'entretient la feuille avec l'écran, les propriétés de l'objet `Screen` et les événements `Load` et `Unload`. Mais la fenêtre Propriétés fournit un moyen plus simple de spécifier la position de la feuille au démarrage : la propriété `StartUpPosition`, qui peut prendre l'une des quatre valeurs présentées dans le Tableau 11.1.

Tableau 11.1 : StartUpPosition permet de spécifier la position initiale de la feuille

Propriété	*Constante nommée*	*Valeur*	*Description*
`0-Manual`	`vbStartUpManual`	`0`	Pas de spécification.
`1-CenterOwner`	`vbStartUpOwner`	`1`	Centre la feuille sur l'écran.
`2-CenterScreen`	`vbStartUpScreen`	`2`	Centre l'élément sur l'écran.
`3-WindowsDefault`	`vbStartUpWindowsDefault`	`3`	Coin supérieur gauche de l'écran.

Les événements de feuilles sont souvent d'une haute importance pour le programme. Vous connaissez déjà `Load` et `Unload`, ainsi que les événements feuille/souris du chapitre précédent. Nous allons maintenant étudier des événements de feuilles tout aussi utiles. Le Tableau 11.2 en présente trois.

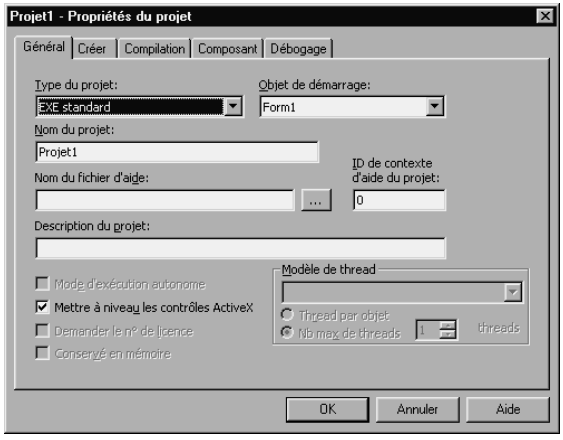
Tableau 11.2 : Trois événements de feuilles

Evénement	*Description*
`Activate`	A lieu lorsque la feuille devient active (parce que l'utilisateur a cliqué ou parce qu'il revient à la feuille en basculant d'une autre application).
`Deactivate`	A lieu lorsqu'une autre feuille — ou une autre application — devient active.
`Resize`	a lieu lorsque l'utilisateur redimensionne la feuille ou lorsque le programme modifie les propriétés `Height` et `Width`.

La propriété `Resize` peut également servir à réorganiser les contrôles après que l'utilisateur a redimensionné la feuille. Si vous affectez aux contrôles une position relative aux propriétés `Height` et `Width` de la feuille, ils resteront centrés quelle que soit la taille de la feuille.

La Figure 11.1 reproduit la boîte de dialogue que l'on obtient par le menu Projet, Propriétés. Cette boîte de dialogue que nous avons déjà évoquée, contient la liste déroulante Objet de démarrage, qui propose les différentes feuilles de l'application ainsi qu'une procédure spéciale nommée `Main`. Certaines applications n'ont pas de feuille de démarrage. Ce peut être le cas d'un utilitaire d'arrière-plan qui n'affiche pas de fenêtre. Il peut aussi s'agir d'une application qui vérifie préalablement une valeur quelconque, afin de déterminer quelle feuille doit apparaître, ou bien qui demande la saisie d'un mot de passe.

Figure 11.1
La boîte de dialogue Propriétés du projet permet de spécifier la feuille ou procédure de démarrage.



Pour qu'un code s'exécute avant tout chargement de feuille, il faut créer une procédure `Main()` (dans la section générale du module de code, et non dans le module de feuille), et sélectionner cette procédure `Main()` comme Objet de démarrage dans les Propriétés du projet. Le code contenu dans `Main()` s'exécutera avant tout autre chose. En fait, aucune feuille ne s'affichera avant que `Main()` ne lui applique la méthode `Show`.

Vous découvrirez d'autres événements liés aux feuilles dans les Chapitres 13 et 14.

Les collections de feuilles

Une feuille, nous l'avons mentionné plus haut, peut être désignée comme *objet*. Visual Basic supporte plusieurs types d'objets : les contrôles, les feuilles et les objets résidant hors de l'application, tels que les objets OLE. (OLE, pour *object linking and embedding*, signifie "liaison et incorporation d'objets". Vous en apprendrez plus à ce sujet aux Chapitres 16 et 17).

L'ensemble de vos objets feuilles constitue la collection `Forms`, qui change selon que vous ajoutez ou supprimez des feuilles dans le projet. La collection `Forms` contient le nom de toutes les feuilles. Par exemple, `frmAboutBox` pourrait être le nom d'une de vos feuilles dans la collection `Forms`. L'objet nommé `Form` (sans "s") définit la feuille actuellement ouverte. C'est à cet objet que Visual Basic se réfère lorsque vous appliquez une méthode de feuille sans spécifier de nom.

La collection `Forms` est l'ensemble des feuilles définies dans votre application.

Accès à la collection *Forms*

Visual Basic vous permet de vous référer aux feuilles de la collection `Forms` sans spécifier le nom des feuilles individuelles ouvertes.

Supposons, par exemple, que vous ayez trois feuilles ouvertes : `frmAcPay`, `frmAcRec` et `frmAcReview`. L'objet `Forms` contient ces trois feuilles. Chaque feuille est indexée, à partir de `0`, et vous pouvez désigner une feuille par ce nombre plutôt que par son nom. Ce nombre, ou *indice*, est spécifié entre parenthèses après le nom de l'objet `Forms`. Le Tableau 11.3 détaille, pour nos trois feuilles d'exemple, les indices correspondants.

La collection `Forms` *fonctionne comme un tableau de variables.*

Tableau 11.3 : Les objets de la collection Forms peuvent être désignés par leur indice

Nom de la feuille	*Notation indiciaire*
`frmAcPay`	`Forms(0)`
`frmAcRec`	`Forms(1)`
`frmAcReview`	`Forms(2)`

Il existe une autre façon de désigner une feuille en employant son nom. Voici un exemple :

```
Forms![frmAcPay]
```

Cet exemple se réfère à une feuille nommée `frmAcPay` *comprise dans la collection courante. (Les crochets sont obligatoires.) Un autre format inclut des parenthèses :*

```
Forms!("frmAcPay")
```

Les indices

On peut également se servir d'un indice à la place du nom pour désigner un contrôle individuel de la feuille, tel qu'une zone de texte.

Imaginons qu'une feuille nommée `frmStore` contient cinq contrôles : trois labels (`lblStoreNum`, `lblStoreName` et `lblStoreLoc`) et deux zones de liste (`lstStoreEmps` et `lstStoreMgrs`). Les procédures peuvent désigner chaque contrôle par son indice, comme le montre le Tableau 11.4. Notez que le nom du contrôle individuel doit être distingué de `Forms` par un point d'exclamation.

Tableau 11.4 : Les contrôles de la feuille peuvent aussi être désignés par leur indice

Nom du contrôle	*Notation indiciaire*
lblStoreNum	`Forms!frmStore(0)`
lblStoreName	`Forms!frmStore(1)`

Tableau 11.4 : Les contrôles de la feuille peuvent aussi être désignés par leur indice *(suite)*

Nom du contrôle	*Notation indiciaire*
lblStoreLoc	`Forms!frmStore(2)`
lstStoreEmps	`Forms!frmStore(3)`
lstStoreMgrs	`Forms!frmStore(4)`

Ne confondez pas les indices désignant un contrôle de la feuille avec les indices de la collection `Forms`. Si `Forms` précède immédiatement l'indice, c'est à une feuille ou sous-feuille précise qu'il est fait référence. S'il suit le nom de la feuille, comme dans le Tableau 11.4, l'indice désigne l'un des contrôles de la feuille.

La propriété *Count*

Comme les contrôles, les collections ont des propriétés. `Count` est l'une des propriétés de la collection `Forms`. La propriété `Count` vous simplifie la tâche en ce qu'elle vérifie pour vous le nombre de feuilles contenues dans une collection. Grâce à `Count`, vous pouvez écrire des procédures généralistes qui s'appliqueront à toutes les feuilles actuellement ouvertes. La propriété `Count` contient toujours un entier.

Appliquée à un nom de contrôle spécifique, la propriété `Count` peut aussi indiquer le nombre de contrôles contenus dans une feuille. Par exemple, `frmAcPay.Count` renvoie le nombre de contrôles contenus dans la feuille `frmAcPay`. `Count` tient compte des contrôles cachés.

Le code suivant déclare une variable `Integer`, `intC`, et y stocke le nombre de feuilles ouvertes :

```
Dim intC As Integer
intC = Forms.Count     ' Stocke le nombre de feuilles ouvertes.
```

`Count` permet aussi de compter les contrôles d'une feuille précise. Le code suivant déclare une variable `Integer`, `intCC`, et y stocke le nombre de contrôles sur une feuille nommée `frmMyForm` :

```
Dim intCC As Integer
intCC = frmMyForm.Count    ' Stocke le nombre de contrôles de
➥la feuille.
```

Placée après le nom d'une feuille, `Count` indique le nombre de contrôles qu'elle contient. Placée après `Forms`, elle indique le nombre de feuilles contenues dans la collection.

Une boucle `For` est l'outil parfait pour parcourir toutes les feuilles du projet en cours. Prenez garde de toujours amorcer la boucle sur une valeur `0`, c'est-à-dire sur l'indice de la première feuille.

Le code suivant masque toutes les feuilles ouvertes :

```
For intI = 0 To Forms.Count - 1
  Forms(intI).Visible = False   ' Masque chaque feuille.
Next intI
```

Il est parfois nécessaire de masquer toutes les feuilles, par exemple pour effectuer une tâche système requérant l'absence totale d'E/S utilisateur. Une fois la tâche effectuée, une boucle semblable permettra de définir la propriété `Visible` de chaque feuille comme `True`.

Visual Basic supporte un format spécial de `For`, `For Each`. Cette boucle parcourt une collection sans qu'il soit besoin de contrôler une variable de boucle. Voici une boucle `For Each` qui masque toutes les feuilles ouvertes :

```
Dim varFrmObj As Variant
For Each varFrmObj In Forms
  varFrmObj.Visible = False   ' Masque chaque feuille.
Next
```

La seule variable de boucle qu'exige `For Each` est une variable `Variant`, qui contiendra le nom de chaque feuille à chaque itération de la boucle dans la collection. Dans le même esprit, l'instruction `For Each` suivante amorce une boucle qui parcourt tous les contrôles de la feuille `frmMyForm`, quel que soit le nombre de ces contrôles :

```
For Each varControl In frmMyForm
```

Déchargement des feuilles

Nous avons déjà évoqué la procédure événementielle `Form_Unload()`. L'événement `Unload` est utile pour les routines de nettoyage, par exemple sauvegarder toutes les données avant que l'application ne se ferme. Mais, surtout, l'événement `Unload` joue un rôle central dans la fermeture "propre" des programmes.

Imaginons une application qui contient plusieurs feuilles. Si l'application masque une feuille, l'utilisateur n'a aucun moyen de savoir que la feuille est encore chargée. D'un autre côté, même si l'utilisateur ferme la feuille principale, celle-ci est toujours

chargée en mémoire et, pour Windows, le programme est toujours en cours d'exécution. Ainsi, le programme continue d'accaparer de la mémoire sans que l'utilisateur en ait connaissance.

Les programmeurs Visual Basic ajoutent souvent le code suivant aux diverses commandes de fermeture (telles que l'option de menu Fichier, Quitter) :

```
For intCtr = (Forms.Count - 1) to 0 Step - 1
  Unload Forms(intCtr)  ' Décharge les feuilles affichées comme
➥les feuilles masquées.
Next intCtr
```

La méthode *Print*

La méthode `Print` permet d'afficher du texte sur la feuille sans passer par un contrôle du type label ou zone de texte. `Print` affiche le texte directement sur la feuille. `Print` a l'inconvénient d'exiger trop de programmation. `Print` n'étant pas un contrôle, on ne peut se contenter de définir des propriétés lors de la création. C'est à travers le code seul que l'on peut régler le comportement de la méthode `Print`. Par exemple, vous devez spécifier dans le code la position précise à laquelle `Print` affichera le texte, sinon elle pourrait se superposer aux contrôles de la feuille.

La méthode `Print` ne permet pas seulement d'envoyer du texte aux feuilles. Comme nous le verrons au Chapitre 13, `Print` est aussi la façon la plus courante d'envoyer une sortie vers l'imprimante.

Plus généralement, `Print` permet d'envoyer du texte à un objet. Pour notre exposé, cet objet sera une feuille. Mais `Print` peut aussi bien s'appliquer aux contrôles Picture, ainsi qu'aux objets `Printer` ou `Debug` (qui est une fenêtre d'exécution spéciale dans laquelle vous pouvez envoyer des résultats de tests lors de l'exécution). L'objet le plus simple auquel puisse s'appliquer la méthode `Print` est la feuille.

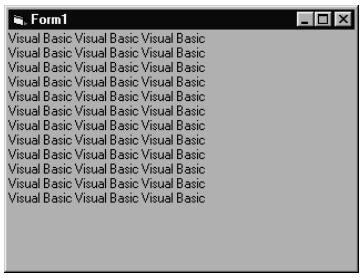
Vous pouvez, dès maintenant, vous exercer à la pratique de `Print`, sans construire une application complète. Ouvrez un nouveau projet, puis double-cliquez sur la fenêtre Form1 pour afficher la fenêtre Code. Puisque le projet vient juste d'être créé, les seuls objets disponibles sont la feuille Form1 et la procédure générale qui contient la section de déclarations. Toutes les procédures prédéfinies que la feuille peut reconnaître sont proposées dans la liste déroulante Procédure. Sélectionnez Click dans cette liste, puis saisissez le code du Listing 11.1. (Comme toujours, Visual Basic insère automatiquement les lignes d'encadrement ; vous ne devez donc taper que le corps de la procédure.)

Listing 11.1 : La méthode Print écrit directement sur la feuille

```
1: Private Sub Form_Click()
2: ' Exemple de méthode Print.
3:   Dim strString As String
4:   strString = "Visual Basic"
5:   ' Affiche trois fois la chaîne.
6:   Form1.Print strString & " " & strString & " " strString
7: End Sub
```

Exécutez le programme. La feuille s'affiche, mais rien ne se passe tant que vous ne cliquez pas dessus. Ce qui est tout à fait normal, puisque le code est inclus dans la sous-routine `Form_Click ()`. Cliquez plusieurs fois. Le résultat devrait ressembler à la Figure 11.2.

Figure 11.2
La méthode Print affiche la sortie directement sur la feuille.



`Print` est l'une des manières les plus simples d'afficher des informations. Pour afficher une chaîne quelconque sur une feuille de votre programme, il suffit de faire précéder `Print` du nom de la feuille, le tout séparé par un point. Voici le format de la méthode `Print` appliquée à une feuille :

```
frmFormName.Print DataToPrint
```

Ici, *`frmFormName`* est la feuille de destination, et *`DataToPrint`* les données à imprimer. `Print` peut envoyer des littéraux (nombres, chaînes ou dates), des valeurs de variables et des contrôles.

Formatage de la sortie *Print*

Vous pouvez formater la sortie de `Print` en incluant les fonctions `Spc()` ou `Tab()`. Ces deux méthodes permettent d'espacer les données envoyées par la méthode `Print`.

Le Listing 11.2 utilise `Spc()` et le point-virgule (`;`) pour envoyer deux chaînes sur la même ligne. `Spc(5)` ordonne à la méthode `Print` d'insérer cinq espaces avant d'afficher

la chaîne sur la sixième colonne. Si vous terminez l'instruction `Print` par un point-virgule, l'instruction `Print` suivante affichera le texte à la suite, sur la même ligne.

Listing 11.2 : Le point-virgule empêche les sauts de lignes

```
1:  Private Sub Form_Click ()
2:    Dim strString As String
3:    strString = "Visual Basic"
4:    Form1.Print "*"; Spc(5); strString;   ' Remarquez le point-virgule.
5:    Form1.Print Spc(2); strString
6:  End Sub
```

Cliquez plusieurs fois sur la feuille, et la sortie suivante s'affichera :

```
*     Visual Basic  Visual Basic
*     Visual Basic  Visual Basic
*     Visual Basic  Visual Basic
```

Le code force `Print` à sauter cinq espaces avant d'afficher le premier `Visual Basic`. Après deux autres espaces, la seconde instruction `Print` affiche également `Visual Basic`. Chaque fois que vous cliquez, la procédure événementielle se répète.

Si l'on emploie `Tab()` au lieu de `Spc()`, la sortie commencera à la colonne spécifiée dans l'argument entre parenthèses. `Spc()` force l'instruction `Print` suivante à sauter un certain nombre d'espaces, alors que `Tab()` ordonne au `Print` suivant de commencer à une colonne précise. Le Listing 11.3 donne un exemple.

Listing 11.3 : Espacement de la sortie de Print à l'aide des fonctions Tab() et Spc()

```
1:  Private Sub Form_Click()
2:    Dim strString As String
3:    strString = "Visual Basic"
4:
5:    Form1.Print "*"; Tab(5); strString; Tab(20); strString
6:    Form1.Print "*"; Spc(5); strString; Spc(20); strString
7: End Sub
```

A la ligne 5, `Tab()` spécifie les colonnes à utiliser, tandis qu'à la ligne 6, `Spc()` spécifie un certain nombre d'espaces à sauter.

Voici la sortie de cette procédure :

```
*    Visual Basic   Visual Basic
*     Visual Basic                    Visual Basic
```

La méthode `Print` peut également servir à insérer des lignes vierges sur une feuille. Examinez le code du Listing 11.4.

Listing 11.4 : Print permet aussi d'insérer des lignes vierges

```
1:  Private Sub Form_Click()
2:    Dim strString As String
3:    Dim CurLine As Integer
4:
5:    CurLine = 1
6:    strString = "Visual Basic"
7:
8:    ' Affiche la ligne.
9:    Form1.Print strString & " est sur la ligne n°" & CurLine
10:
11:   For CurLine = 2 To 6
12:     Form1.Print     ' Insère des lignes vierges.
13:   Next CurLine
14:
15:   ' Affiche la ligne.
16:   Form1.Print strString & " est sur la ligne n°" & CurLine
17: End Sub
```

La sortie présente cinq lignes vierges entre les deux chaînes :

```
Visual Basic est sur la ligne n°1




Visual Basic est sur la ligne n°7
```

Les lignes 11 à 13 insèrent des lignes vierges parce que la méthode `Print` n'envoie pas de données.

Positionnement de la sortie *Print*

Il faut parfois spécifier la position exacte où la sortie doit s'afficher. A cette fin, plusieurs propriétés liées à la feuille peuvent être utilisées conjointement à la méthode `Print`. Ces propriétés renvoient la position courante du curseur texte, qui se déplace au fur et à mesure de l'exécution de la méthode `Print`. Les coordonnées du curseur texte sont contenues dans les propriétés `CurrentX` et `CurrentY`. Vous pouvez, à l'aide de ces propriétés, définir avec précision le point où les données seront affichées.

Le comportement de `CurrentX` et `CurrentY` est affecté par une troisième propriété, `ScaleMode`. La feuille peut reconnaître plusieurs modes différents selon le contenu de la propriété `ScaleMode`. Le mode en question concerne l'échelle employée pour l'affichage des images et du texte sur la feuille. Le Tableau 11.5 présente les divers modes disponibles.

La plupart sont de nature graphique et particulièrement utiles pour les sorties imprimantes (nous en reparlerons au Chapitre 13).

Tableau 11.5 : Les valeurs de ScaleMode déterminent les coordonnées de la sortie Print

Constante nommée	*Valeur*	*Echelle*
`vbUser`	`0`	Spécifiée par le programmeur.
`vbTwips`	`1`	Echelle en twips (valeur par défaut).
`vbPoints`	`2`	Point typographique (environ 0,35 mm).
`vbPixels`	`3`	Point le plus petit de l'objet (pour l'écran, la taille du point d'écran détermine celle du pixel).
`vbCharacters`	`4`	Taille d'un caractère.
`vbInches`	`5`	Un pouce (25,4 mm).
`vbMillimeters`	`6`	1 mm.
`vbCentimeters`	`7`	1 cm.
`vbHimetric`	`8`	Certains programmeurs préfèrent aux pixels les mesures *Himetric*, indépendantes du matériel ; Windows utilise alors la plus haute résolution possible.
`vbContainerPosition`	`9`	Utilisée comme valeur positionnelle, la valeur `ScaleMode` du conteneur (un objet qui contient l'objet courant) détermine la valeur `ScaleMode` de l'objet courant.
`vbContainerSize`	`10`	Utilisée comme valeur positionnelle, la valeur `ScaleMode` du conteneur détermine la valeur `ScaleMode` de l'objet courant.

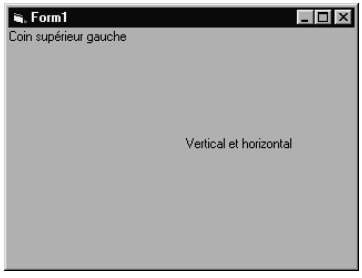
Pour le texte, la valeur de `ScaleMode` la plus courante est `vbCharacters`. Ainsi, si `CurrentX` et `CurrentY` ont toutes les deux la valeur `5`, la prochaine sortie `Print` commencera colonne 5, ligne 5. L'origine pour la propriété `ScaleMode` est le coin supérieur gauche de la feuille (coordonnées `0, 0`). La procédure événementielle `Click` du Listing 11.5 exploite les propriétés `ScaleMode`, `CurrentX` et `CurrentY`.

Listing 11.5 : Positionnement de la sortie à l'aide des propriétés CurrentX et CurrentY

```
1:  Private Sub Form_Click()
2:     ' Définition de l'echelle
3:     Form1.ScaleMode = VbCharacters
4:
5:     Form1.CurrentX = 20   ' Déplacement horizontal de 20 caractères.
6:     Form1.CurrentY = 6    ' Déplacement vertical de 6 lignes.
7:     Form1.Print "Vertical et horizontal "
8:
9:     Form1.CurrentX = 0   ' Retour vers la gauche.
10:    Form1.CurrentY = 0   ' Retour vers le haut.
11:    Form1.Print "Coin supérieur gauche"
12: End Sub
```

La ligne 3 affecte à `ScaleMode` la constante `VbCharacters` qui spécifie l'échelle caractère. La sortie du Listing est reproduite en Figure 11.3. Vous pouvez remarquer que la sortie du second `Print` s'affiche plus haut sur la feuille que la sortie du premier.

Figure 11.3
CurrentX et CurrentY positionnent le curseur texte de la méthode Print.



Création de nouvelles propriétés de feuilles

Vous pouvez créer vos propres propriétés de feuilles. Il vous arrivera sans doute, pour les besoins d'une application, d'appliquer à une feuille les mêmes modifications à plusieurs reprises. Dans un cas pareil, il peut être intéressant de définir des propriétés personnalisées.

Imaginons, par exemple, que vouliez afficher un titre au bas d'une feuille. Si vous vous servez d'un label, il faudra configurer le contrôle, et le configurer de nouveau chaque fois que vous voudrez modifier le titre affiché. En faisant de ce titre l'une des propriétés de la feuille, vous vous épargnez une peine inutile.

Pour bien comprendre la création de propriétés personnalisées, rien de mieux qu'un exemple. Créez un nouveau projet et configurez-le selon les spécifications du Tableau 11.6. Cette petite application montre comment les valeurs de `ScaleMode` déterminent les coordonnées de la sortie `Print`.

Tableau 11.6 : Les coordonnées de Print sont modifiées par les valeurs de ScaleMode

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmTitle`
Feuille	`Caption`	`Titre`
Feuille	`Height`	`3720`
Feuille	`StartUpPosition`	`2-CenterScreen`
Feuille	`Width`	`3975`
Bouton de commande	`Name`	`cmdTitle`
Bouton de commande	`Caption`	`&Voir le titre`
Bouton de commande	`Height`	`495`
Bouton de commande	`Left`	`1320`
Bouton de commande	`Top`	`1200`
Bouton de commande	`Width`	`1215`

Lorsque l'utilisateur clique sur le bouton de commande, le titre s'affiche au bas de la feuille. Cela est dû à la procédure événementielle `Click` du bouton de commande :

```
1:  Private Sub cmdTitle_Click()
2:    frmTitle.BottomTitle = "Nouvelle propriété"
3:  End Sub
```

Examinez la ligne 2. Il y a quelque chose qui cloche : il n'existe pas de propriété nommée `BottomTitle`. Qu'à cela ne tienne, nous allons la créer !

Suivez ces étapes pour ajouter la nouvelle propriété `BottomTitle` à la liste des propriétés de votre feuille :

1. Cliquez sur le bouton Code de la fenêtre Propriétés pour afficher la fenêtre Code.

2. Dans la section de déclarations qui apparaît avant la procédure événementielle `cmdTitle_Click()`, tapez ceci :

```
Dim strTitle As String
```

`strTitle` est donc une variable publique, disponible pour le projet entier. Puisque les variables publiques sont tant décriées, pourquoi déclarer `strTitle` comme publique ? Rappelez-vous que les contrôles sont publics pour l'application entière. Il n'y a pas de contrôles privés, pas plus que de propriétés privées. La variable `strTitle` contient en fait la valeur de la nouvelle propriété `BottomTitle` qui ne deviendra réellement propriété qu'à la prochaine étape. Or cette propriété `BottomTitle` doit avoir un accès complet et permanent à sa variable réservée `strTitle`. Il faut donc que cette variable soit publique.

3. Vous allez maintenant découvrir un nouveau type de procédure : `Property Get`. Il existe une procédure `Property Get` pour chaque propriété que vous définissez. Lorsque le programme rencontre une propriété personnalisée, Visual Basic exécute automatiquement la procédure `Property Get` correspondante. Ainsi, après que l'on aura créé la procédure `Property Get` de la propriété `BottomTitle`, Visual Basic exécutera automatiquement cette procédure chaque fois que l'application demandera la valeur de `BottomTitle`.

Une procédure `Property Get` renvoie la valeur de la propriété créée.

Dans la fenêtre Code, saisissez le contenu du Listing 11.6.

Listing 11.6 : La procédure Property Get renvoie la valeur de la propriété

```
1:  Public Property Get BottomTitle()
2:     ' Cette procédure renvoie la valeur
3:     ' de la propriété BottomTitle,
4:     ' valeur en fait contenue dans
5:     ' la variable publique strTitle.
6:      BottomTitle = strTitle
7:  End Property
```

La propriété `BottomTitle` n'est pas vraiment une propriété. Sa valeur reste contenue dans la variable publiques `trTitle`. Mais, grâce à la procédure `Property Get`, le programme n'y voit que du feu et traite `BottomTitle` comme une propriété normale. En vérité, la valeur de la propriété est une valeur de variable.

4. Il faut maintenant fournir à notre pseudo-propriété `BottomTitle` une procédure `Property Let`. La procédure `Property Let` change la valeur de `BottomTitle`. En fait, `Property Let` n'intervient que sur la variable publique qui "représente" `BottomTitle` dans le code, et à laquelle sont affectées les valeurs destinées à cette propriété.

Une procédure `Property Let` affecte une valeur à la propriété créée.

Dans la fenêtre Code, saisissez le contenu du Listing 11.7.

Listing 11.7 : La procédure Property Let affecte une valeur à la propriété

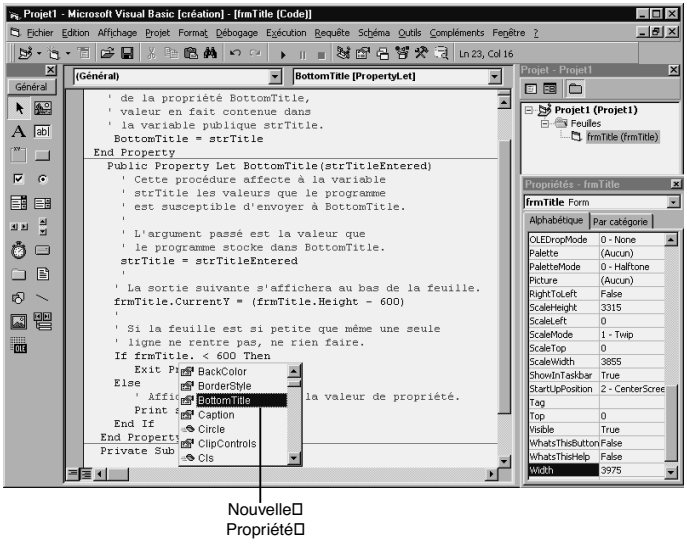
```
1:  Public Property Let BottomTitle(strTitleEntered)
2:    ' Cette procédure affecte à la variable
3:    ' strTitle les valeurs que le programme
4:    ' est susceptible d'envoyer à BottomTitle.
5:    '
6:    ' L'argument passé est la valeur que
7:    ' le programme stocke dans BottomTitle.
8:    strTitle = strTitleEntered
9:    '
10:   ' La sortie suivante s'affichera au bas de la feuille.
11:   frmTitle.CurrentY = (frmTitle.Height - 600)
12:   '
13:   ' Si la feuille est si petite que même une seule
14:   ' ligne ne rentre pas, ne rien faire.
15:   If frmTitle.CurrentY < 600 Then
16:      Exit Property
17:   Else
18:      ' Affiche sur la feuille la valeur de propriété.
19:      Print strTitle
20:   End If
21: End Property
```

La ligne 8 affecte à la variable publique l'argument passé à la procédure. Quel que soit le point du programme où une valeur est affectée à la propriété `BottomTitle`, cette procédure s'exécute pour passer la valeur comme argument.

Les lignes 11 et 15 règlent l'affichage du titre. La ligne 11 fait en sorte que le libellé s'affiche à 600 twips du bas de la feuille, quelle que soit la taille définie en cours d'exécution par l'utilisateur. Si l'utilisateur a rétréci la fenêtre au point que même une seule ligne ne puisse apparaître, la ligne 16 annule toute la procédure. Tant que la ligne de texte rentre sur la feuille, la ligne 19 affiche la valeur à la position `CurrentY`. (`CurrentX` vaut `0`, soit le bord gauche de la fenêtre. Ici, le code n'a jamais modifié `CurrentX`.)

Dès lors que vous avez écrit les procédures `Property Get` *et* `Property Let`*, la propriété* `BottomTitle` *fait partie intégrante de la liste des propriétés de la feuille. Bien que la fenêtre Propriétés n'inclura pas toujours la nouvelle propriété, ce sera le cas de toutes les autres listes de propriétés. Dans la Figure 11.4, on s'apprête à affecter une valeur à l'une des propriétés de la feuille ; le menu contextuel qui s'affiche propose bien la propriété* `BottomTitle`*. La nouvelle propriété peut être utilisée comme les autres, par la grâce des procédures* `Property Get` *et* `Property Let`*.*

Figure 11.4
La propriété BottomTitle apparaît dans le menu contextuel de la fenêtre Code.



5. Lorsque vous lancez l'application et cliquez sur le bouton de commande, l'événement `Click` de ce contrôle se produit, et la procédure événementielle `Click` s'exécute. La procédure événementielle `Click` affecte le littéral chaîne `"Nouvelle propriété"`, la propriété `BottomTitle`, et les procédures `Property` s'occupent du reste.

Les applications multifeuilles

La plupart des applications que nous avons créées jusqu'ici ne contenaient qu'une seule feuille. Nous n'avons pas abordé la question des applications multifeuilles, en raison de la simplicité des projets étudiés. Pour ajouter une feuille, il suffit de cliquer du bouton droit sur la fenêtre Projet, puis choisir l'option appropriée.

Vous allez maintenant apprendre à travailler sur des applications multifeuilles. Les feuilles ajoutées seront des feuilles de données spéciales. Mais avant toute chose, vous devez saisir la distinction entre les programmes SDI (*Single Document Interface*, interface monodocument) et MDI (*Multiple Document Interface*, interface multidocument).

Lorsque plusieurs feuilles sont en jeu, on doit généralement s'occuper de plusieurs jeux de contrôles. En assignant à chaque feuille un nom assez explicite, vous réduisez la part de complications que cela implique. Le nom de la feuille permet à tout moment de savoir sur quel jeu de contrôles on travaille. Du reste, les feuilles peuvent être affichées et masquées à volonté lors de l'exécution (l'utilisateur peut aussi basculer de lui-même d'une feuille à l'autre).

Même lorsque plusieurs feuilles sont affichées, il n'y a qu'une feuille active. Pour activer une feuille inactive, il suffit à l'utilisateur de cliquer sur l'une de ses parties visibles. Une feuille peut aussi être activée depuis le code au moment choisi. La méthode `frmForm.Show` *active la feuille* `frmForm` *et masque les autres feuilles si la feuille activée manque d'espace.*

Les applications développées en mode MDI peuvent devenir assez complexes. La plupart des applications commerciales sont de type multidocument. Les programmes décrits dans la suite de cet ouvrage, plus puissants et exploitant des fichiers et des contrôles supplémentaires, seront presque toujours des applications MDI.

Plutôt que de cliquer du bouton droit sur la fenêtre Projet pour ajouter une feuille, vous pouvez aussi choisir, dans le menu Projet, la commande Ajouter une feuille. Visual Basic affiche alors la boîte de dialogue Ajouter une feuille (voir Figure 11.5). Vous pouvez choisir entre plusieurs types de feuilles, ou choisir dans la liste proposée à l'onglet Existant.

Si vous double-cliquez sur l'icône Form, Visual Basic ouvre une nouvelle feuille standard. Chaque feuille ajoutée est automatiquement nommée Form2, Form3, etc. Il convient de donner aux feuilles des noms plus explicites dès la première étape de la création.

Le Chapitre 15 détaille les autres options proposées dans la boîte de dialogue Ajouter une feuille.

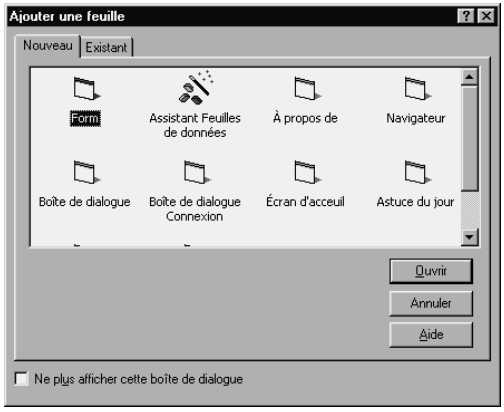


Figure 11.5
Sélectionnez le type de feuille à ajouter.

SDI contre MDI

Visual Basic supporte trois styles d'interface :

- **Interface monodocument (SDI).** Une application SDI contient une seule fenêtre de données. Dans l'utilitaire Windows Bloc-notes, par exemple, on ne peut ouvrir qu'un seul document à la fois. La barre de menus des applications SDI ne contient généralement pas de menu Fenêtre, puisque l'on ne peut passer d'une fenêtre à l'autre. Dès que l'utilisateur ouvre un nouveau fichier de données, ce contenu vient prendre la place de la fenêtre précédemment ouverte.
- **Interface multidocument (MDI).** Une application MDI contient plusieurs fenêtres de données (*fenêtres de document*). Microsoft Word, par exemple, permet d'ouvrir autant de documents que l'on veut. Le code MDI de Word garantit que chaque document apparaît dans sa propre fenêtre (voir Figure 11.6).

  On passe de l'une à l'autre en cliquant sur la fenêtre appropriée, ou en la sélectionnant dans le menu Fenêtre. Lorsque l'utilisateur bascule entre les documents — c'est-à-dire, de notre point de vue, entre les feuilles —, la fenêtre sélectionnée reçoit le focus et devient la feuille active.
- **Interface Explorateur.** C'est le style mis en œuvre dans le système d'aide Visual Basic et dans l'Explorateur Windows. Une application de type Explorateur contient deux fenêtres : une à gauche et une à droite. La fenêtre de gauche affiche une vue arborescente des données détaillées dans la fenêtre de droite. L'interface Explorateur est appropriée aux applications de gestion de fichiers et d'images. On utilise pour les applications de ce type les contrôles TreeView et ListView, qui permettent de parcourir les données des deux panneaux. (Ces contrôles n'apparaissant pas par défaut sur la Boîte à outils, il faut les ajouter par le menu Projet, Composants.)

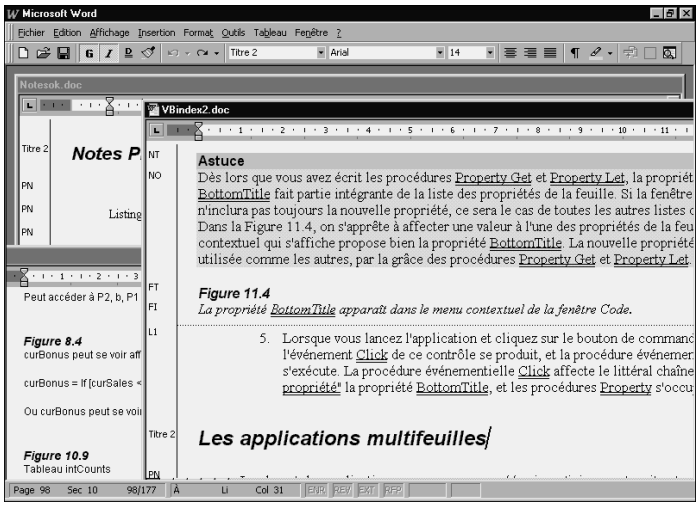


Figure 11.6
Les applications MDI contiennent de multiples fenêtres de données.

Info

Une application SDI peut contenir plusieurs feuilles. MDI signifie simplement que l'application peut inclure une ou plusieurs feuilles filles, chacune contenant un jeu de données particulier. Dans une application MDI, les diverses feuilles apparaissent dans une feuille de contrôle (on parle aussi de feuille parent ou de feuille principale), et ne peuvent s'afficher que dans les limites de cette feuille de contrôle. Une application SDI peut contenir de multiples feuilles, mais aucune n'est considérée comme la fille d'une autre. Si votre application est destinée à traiter un seul jeu de données à la fois (un seul fichier client ou le fichier paye d'un seul employé, par exemple), ou si elle ne traite pas de données en dehors des informations de contrôle du programme, alors le mode SDI est le plus indiqué.

Terminologie MDI

Pour exploiter adéquatement le mode MDI, vous devez en connaître la terminologie. La feuille principale qui fait office de toile de fond pour les autres feuilles, est souvent désignée comme *feuille parent* ou *fenêtre parent*. La fenêtre parent est le réceptacle qui contient une ou plusieurs fenêtres filles. Dans Microsoft Word, par exemple, l'arrière-plan gris, avec ses barres de menus, d'état et d'outils, est la fenêtre parent. Chaque document Word ouvert s'affiche dans une fenêtre fille à l'intérieur de la fenêtre parent ; les fenêtres filles ne peuvent sortir des limites de la fenêtre parent.

Cette fenêtre parent qui impose ses limites aux fenêtres filles ne peut contenir que deux types de contrôles :

- les contrôles qui supportent la propriété `Align` ;
- les contrôles sans interface visible (voir Chapitre 15).

Si l'utilisateur réduit une fenêtre fille, l'icône ou le bouton correspondant s'affiche au bas de la fenêtre parent, mais pas sur la Barre des tâches. Les fenêtres filles ne peuvent jamais dépasser les frontières de la fenêtre parent ; les limites de la fenêtre parent sont les limites absolues de toutes les fenêtres filles.

Techniquement, une fenêtre fille se distingue par ce que la propriété `MDIChild` de la feuille correspondante est définie comme `True`. Du reste, une application MDI peut contenir des feuilles non-filles. Ainsi, par exemple, de la boîte de dialogue A propos (accessible depuis le menu d'aide) ; cette boîte de dialogue ne contenant pas de données proprement dites, elle n'a pas à être une fille MDI.

Pour créer une application MDI, choisissez dans le menu Projet l'option Ajouter une feuille MDI, puis définissez comme `True` la propriété `MDIChild` de la nouvelle feuille.

L'assistant Création d'applications

L'assistant Création d'applications que nous avons découvert au Chapitre 1, peut également servir à créer des applications MDI. C'est une méthode plus simple que de créer les différentes feuilles et de définir chaque propriété `MDIChild` "à la main".

La deuxième boîte de dialogue affichée par l'assistant Création d'applications vous propose de choisir un style d'interface (voir Figure 11.7).

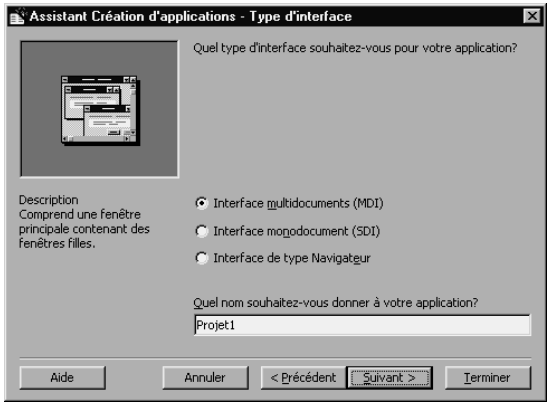


Figure 11.7
L'assistant Création d'applications propose les trois styles d'interface.

Dans les projets MDI construits par l'assistant Création d'applications, les multiples fenêtres filles sont créées à partir de la commande Nouveau du menu Fichier. Lorsque l'utilisateur sélectionne la commande Fichier, Nouveau (procédure événementielle `mnuFileNew .Click()`), le code du Listing 11.8 s'exécute.

Listing 11.8 : Création d'une variable de référence pointant vers la fenêtre fille

```
1:  Private Sub LoadNewDoc()
2:     Static lDocumentCount As Long
3:     Dim frmD As frmDocument
4:
5:     lDocumentCount = lDocumentCount + 1
6:     Set frmD = New frmDocument
7:     frmD.Caption = "Document " & lDocumentCount
8:     frmD.Show
9:  End Sub
```

Ce code est un peu complexe, mais vous en savez déjà assez pour le comprendre. A la ligne 2, la variable statique `lDocumentCount` est locale pour la procédure, mais ne sera jamais à cours de portée. A la première exécution de cette procédure, la valeur de `lDocumentCount` est `0` (les variables statiques commencent toujours par là). Si la procédure change la valeur de `lDocumentCount` (elle lui ajoute 1 à chaque exécution, ligne 5), Visual Basic "retient" la nouvelle valeur. Aucune autre procédure ne peut accéder à `lDocumentCount`, et sa valeur reste toujours identique dans cette procédure-ci. La valeur reste la même, ce qui ne serait pas le cas pour une variable locale *automatique*, soit le contraire d'une variable *statique*. (Toutes les variables que nous avons jusqu'ici déclarées localement étaient des variables automatiques.)

Une variable statique conserve sa valeur même après la fin de la procédure qui la contient. Ainsi, une variable dont la valeur serait `7` au terme d'une procédure, garderait cette valeur même si la procédure s'exécutait de nouveau (par exemple au sein de la boucle d'une autre procédure).

La ligne 3 introduit une instruction d'affectation d'un nouveau genre. En effet, plutôt qu'une variable, c'est une feuille que l'on déclare. L'application contiendra au démarrage la feuille fille `frmDocument` spécifiée lors de la création. L'instruction `Dim` de la ligne 3 déclare une variable `frmD` qui correspond en fait au même objet que `frmDocument`. Au lieu de contenir des données `Integer` ou `String`, `frmD` pointe vers un objet document qui a les mêmes propriétés que `frmDocument`. `frmD` est un exemple de contrôle de variable.

Après que la variable statique chargée de référencer les nouveaux documents a été mise à jour (ligne 5), l'instruction `Set` crée un nouveau document et définit `frmD` comme référence de ce document. En pratique, `frmD` *est* le nouveau document. L'instruction

suivante affecte à la propriété `Caption` du nouveau document la valeur `Document` suivie du numéro de document. Enfin, la dernière instruction applique la méthode `Show` pour afficher le nouveau document dans la fenêtre parent.

L'instruction `Set` fonctionne en gros comme une instruction d'affectation, à ceci près qu'elle affecte une variable de contrôle à un contrôle proprement dit. La ligne 3 du Listing 11.8 ne déclare pas une nouvelle feuille, mais bien une variable de feuille. Cette variable reste vide jusqu'à ce qu'une instruction `Set` l'associe à une feuille déterminée (ce qui est fait ligne 6).

Chaque fois que l'utilisateur choisit Fichier, Nouveau, la procédure `LoadNewDoc()` s'exécute et crée une nouvelle fenêtre fille. Si, dans l'application MDI créée à l'aide de l'assistant Création d'applications, vous sélectionnez Fichier, Fermer, rien ne se passe. L'assistant ne génère pas de code à cet effet. C'est à vous de programmer le déchargement de la fenêtre fille active.

Les fenêtres filles font de parfaites boîtes de dialogue. Le Chapitre 9 vous a expliqué comment générer des boîtes de dialogue communes. Mais vous pouvez créer vos propres boîtes de dialogue, sous la forme de fenêtres filles équipées des contrôles adéquats. Par exemple, un bouton de commande défini comme bouton par défaut via la propriété `Default`, et des zones de texte affichant des entrées par défaut. Les boutons de commande peuvent également être désactivés (propriété `Enabled` définie comme `False`), selon le contexte dans lequel l'utilisateur invoque la boîte de dialogue. Pour afficher une telle boîte de dialogue, il suffit d'appliquer la méthode `Show` :

```
frmDialog.Show vbModal, Me
```

`vbModal` génère une boîte de dialogue modale à laquelle l'utilisateur doit répondre par OK ou Annuler avant de faire quoi que ce soit d'autre dans l'application. La deuxième option de `Show` correspond au nom de la feuille parent, mais vous pouvez employer `Me` si la feuille parent est la feuille standard de l'application.

Les barres d'outils

Les barres d'outils permettent d'accéder par des boutons aux commandes et options de menu courantes. Pour intégrer une barre d'outils à votre application, il faut d'abord ajouter le contrôle Toolbar à votre Boîte à outils. Une fois la barre d'outils créée, il ne reste qu'à écrire pour chaque bouton une procédure événementielle `Click`, comme on le fait pour les options de menu.

Dans une application MDI, les barres d'outils peuvent apparaître sur la feuille parent seule, ou sur les feuilles filles également.

Ajout du contrôle Toolbar à la Boîte à outils

Les barres d'outils et les coolbars (que vous découvrirez dans la prochaine section) fonctionnent et s'installent de la même manière. Si les barres d'outils sont les plus courantes (on en trouve dans à peu près dans toutes les applications Windows d'aujourd'hui), les coolbars sont un peu plus simples à ajouter, car il y a moins de valeurs de propriétés à définir. Cette section vous indique les étapes à suivre pour ajouter une barre d'outils à votre application. La section suivante traite des contrôles spécifiques aux coolbars.

Comme l'indique la section suivante, les coolbars ont ceci de particulier qu'elles permettent de faire glisser les contrôles hors du champ de vision, puis de les faire réapparaître.

Pour ajouter le contrôle Toolbar à votre Boîte à outils, choisissez Projet, Composants. Dans la boîte de dialogue Composants qui s'affiche, sélectionnez l'entrée Microsoft Windows Common Controls 6.0, puis cliquez sur OK. Plusieurs nouveaux outils apparaissent alors dans la fenêtre Boîte à outils. Positionnez le pointeur de la souris sur chacun d'eux pour en connaître le nom.

Pour créer des barres d'outils, voici la procédure générale à suivre :

1. Double-cliquez sur le contrôle Toolbar de la Boîte à outils pour insérer une barre d'outils en faut de la feuille. (Pour que le contour de la barre d'outils soit visible, l'affichage de la grille doit être activé.) La barre d'outils s'étend automatiquement sur toute la largeur de la feuille, quelle que soit cette largeur. Ce qui vous épargne au moins le réglage des dimensions.
2. Pour que les boutons de votre barre d'outils puissent présenter des images, et non seulement du texte, il faut aussi ajouter à la feuille le contrôle ImageList. ImageList fait partie des contrôles conjoints qui viennent avec Toolbar lorsque vous ajoutez l'outil Microsoft Windows Custom Controls 6.0. Le contrôle ImageList contiendra les images que vous disposerez sur la barre d'outils. La disposition du contrôle ImageList sur la barre d'outils est indifférente. Ce contrôle n'apparaît pas sur la feuille lors de l'exécution. Sa seule fonction est d'afficher les icônes des différents boutons de la barre d'outils.

Bien que l'emplacement par défaut — et le plus courant — de la barre d'outils soit le sommet de la feuille, vous pouvez l'afficher en bas, en affectant à la propriété `Align` *du contrôle Toolbar la valeur* `vbAlignBottom`.

3. Dans la fenêtre Propriétés du contrôle ImageList, double-cliquez sur `(Personnalisé)`. La boîte de dialogue Pages de propriétés s'affiche (voir Figure 11.8).

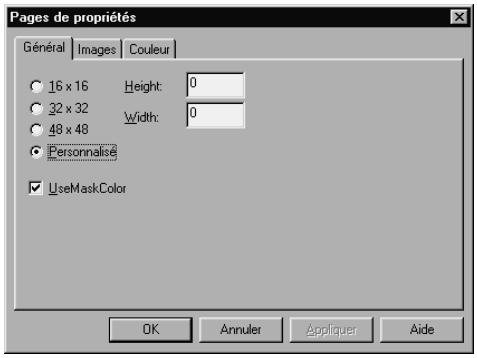


Figure 11.8
La boîte de dialogue Pages de propriétés facilite le réglage des propriétés du contrôle ImageList.

4. Cliquez sur l'onglet Images.
5. Cliquez sur le bouton Insérer une image et sélectionnez une icône.
6. A mesure que vous ajoutez des icônes, Visual Basic actualise les valeurs `Index` de chacune. Ces valeurs d'index permettront d'associer les icônes aux différents boutons de la barre d'outils.
7. Fermez la boîte de dialogue Pages de propriétés.
8. Dans la fenêtre Propriétés du contrôle Toolbar, double-cliquez sur `(Personnalisé)`. La boîte de dialogue Pages de propriétés s'affiche (voir Figure 11.9). Comme pour le contrôle ImageList, la boîte de dialogue Pages de propriétés facilite le réglage des propriétés.
9. Définissez les différentes valeurs de propriétés, puis cliquez sur Appliquer pour constater les changements apportés à la barre d'outils. Dans les Pages de propriétés du contrôle Toolbar, assurez-vous que l'option `ImageList` corresponde bien à votre contrôle ImageList. A l'onglet Boutons, l'option `ToolTipText` permet d'affecter des info-bulles aux boutons.
10. Cliquez sur OK pour refermer les Pages de propriétés du contrôle Toolbar.
11. Affectez à chaque bouton de la barre d'outils une procédure événementielle `Click`.

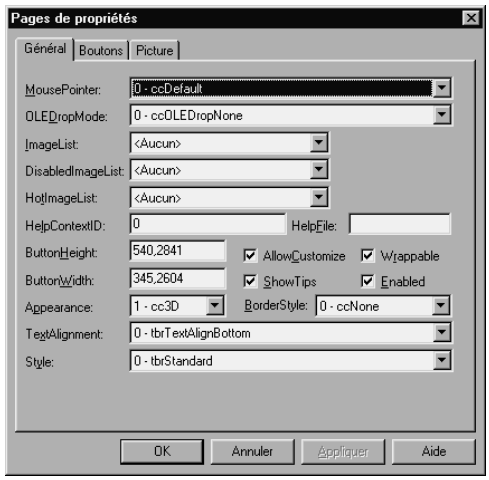


Figure 11.9
Le contrôle Toolbar a lui aussi ses Pages de propriétés.

Définie comme `True`, la propriété `Wrappable` permet à la barre d'outils de s'étendre sur une ligne supplémentaire pour pouvoir, si nécessaire, afficher toutes les icônes.

Vous voulez gâter vos utilisateurs ? Permettez-leur de personnaliser la barre d'outils, en définissant comme `True` la propriété `AllowCustomize` du contrôle Toolbar. A l'exécution, un double-clic sur la barre d'outils affichera la barre d'outils reproduite en Figure 11.10.

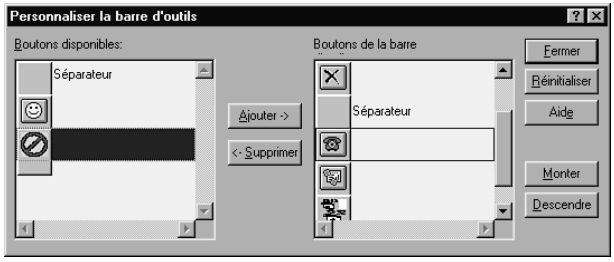


Figure 11.10
Vous pouvez autoriser l'utilisateur à personnaliser la barre d'outils.

Les coolbars

Les coolbars sont une nouveauté de la version 6 de Visual Basic. La Figure 11.11 montre une coolbar Internet Explorer. Grâce à une "poignée" spéciale, l'utilisateur peut faire coulisser la barre vers la droite comme vers la gauche. Les coolbars vous permettent d'inclure sur une même barre une profusion de boutons, que l'utilisateur masque ou affiche à loisir en faisant glisser la poignée.

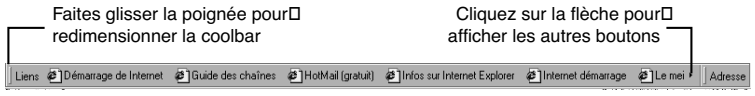


Figure 11.11
La coolbar, coulissante, permet de masquer et d'afficher les icônes à volonté.

Pour pouvoir utiliser le contrôle Coolbar, vous devez ajouter à la Boîte à outils l'outil Microsoft Windows Custom Controls-3 6.0.

Pour ajouter une coolbar à la feuille, la procédure est la même que pour la barre d'outils. Il faut ajouter le contrôle ImageList, puis associer chaque image à la coolbar, à l'aide de la valeur `Key`. L'installation des coolbars est plus simple, car elle exige moins de réglages de propriétés.

Pour ajouter une coolbar, suivez ces étapes :

1. Pour ajouter le contrôle ImageList à votre Boîte à outils, choisissez Projet, Composants, puis cochez l'option Microsoft Windows Common Controls 6.0. Pendant que vous y êtes, cochez l'option Microsoft Windows Common Controls-3 6.0, afin d'ajouter également le contrôle Coolbar. Cliquez sur OK. Les contrôles ImageList et Coolbar apparaissent sur votre Boîte à outils.
2. Double-cliquez sur le contrôle ImageList pour ajouter une liste d'images. La position du contrôle sur la barre n'a aucune importance, puisque ce contrôle n'apparaîtra pas sur la feuille lors de l'exécution. (Sa seule fonction sera d'afficher les icônes sur la coolbar.)
3. Double-cliquez sur la propriété `(Personnalisé)` du contrôle ImageList pour afficher la boîte de dialogue Pages de propriétés.
4. Cliquez sur l'onglet Images.

5. Cliquez sur le bouton insérer une image. La boîte de dialogue Sélectionner un dessin permet de spécifier le chemin d'accès du fichier graphique qui servira de première icône.
6. Sélectionnez, par exemple, une image bitmap du dossier Graphics de Visual Basic. Lorsque vous double-cliquez sur le fichier, la boîte de dialogue se referme, et l'image apparaît au début de la liste (voir Figure 11.12).

Figure 11.12
La première image apparaît dans la liste ImageList.

7. Continuez à choisir vos images.
8. Double-cliquez sur le contrôle Coolbar pour ajouter les deux bandes coulissantes de la coolbar sur votre feuille.
9. Double-cliquez sur la propriété `(Personnalisé)` pour afficher les Pages de propriétés du contrôle Coolbar (voir Figure 11.13).
10. Dans la zone `ImageList`, sélectionnez la liste d'images appropriée. Le Tableau 11.6 décrit les autres options proposées à la page Général de la boîte de dialogue.
11. Cliquez sur l'onglet Bands pour ajouter les icônes.
12. La page propose également les boutons Insérer une bande et Supprimer la bande. Les autres options sont détaillées dans le Tableau 11.7.
13. La propriété `Image` est capitale pour l'ImageList de la coolbar. Chaque valeur `Image` correspond à un élément de la liste d'images. Ainsi, pour une liste d'images contenant dix icônes, vous devrez affecter dix valeurs à `Image`.

Figure 11.13
Les Pages de propriétés facilitent le paramétrage de la coolbar.

Tableau 11.6 : Options de la page Général pour le contrôle Coolbar

Option	*Description*
`Orientation`	Dispose la coolbar à l'horizontale (option par défaut) ou à la verticale. Les coolbars sont généralement horizontales.
`Picture`	Spécifie l'image qui apparaîtra sur les boutons de la coolbar. En général, on se sert plutôt du contrôle ImageList, car l'option `Picture` affecte une *même* image à tous les boutons. L'option `Picture` permet également de régler d'autres aspects de la coolbar, par exemple la "réaction graphique" d'un bouton lorsqu'on le sélectionne.
`ForeColor`	Spécifie la couleur de premier plan.
`BackColor`	Spécifie la couleur d'arrière-plan.
`BandBorders`	Spécifie si des lignes apparaîtront pour séparer les bandes, dans le cas d'une coolbar à plusieurs bandes.
`FixedOrder`	Spécifie si l'utilisateur pourra modifier l'ordre des bandes.
`VariantHeight`	Spécifie si toutes les bandes auront la même hauteur, ou si la hauteur de chaque bande doit dépendre du bouton le plus grand.

Vous ne verrez pas les images sur la coolbar après avoir fermé la boîte de dialogue Pages de propriétés. Les images ne s'affichent qu'à l'exécution.

Tableau 11.7 : Options de la page Bands pour le contrôle Coolbar

Option	*Description*
`Child`	Dans un contexte de programmation avancée, spécifie si un contrôle fils (un contrôle autre qu'un bouton de commande) pourra apparaître sur la coolbar.
`Style`	Spécifie si l'utilisateur pourra redimensionner la coolbar : oui si la valeur est `cc3BandNormal`, non si la valeur est `cc3BandFixedSize` (aucune poignée).
`UseCoolbarPicture`	Détermine si la coolbar affichera les images spécifiées par la propriété `Picture` de l'onglet Bands ou les images spécifiées par le contrôle ImageList.
`Picture`	Spécifie les images qui s'afficheront sur la coolbar, à moins que la propriété `UseCoolbarPicture` n'en décide autrement.
`Caption`	Spécifie le libellé qui apparaîtra sous l'image.
`Width, MinWidth, MinHeight`	Spécifient la taille, en twips, de la coolbar.
`Key`	Disponibles pour les coolbars utilisées en collection. On peut accéder aux boutons par l'indice de collection ou bien par une valeur `Key` unique.
`Tag`	Contient des informations relatives à la coolbar, destinées au programmeur.
`EmbossPicture`	Si la valeur est `False`, l'image apparaît dans ses couleurs originales. Si la valeur est `True`, l'image s'affiche selon les couleurs de premier et d'arrière-plan spécifiées par les propriétés `EmbossHighlight` et `EmbossShadow`.
`UseCoolbarColors`	Si la valeur est `True`, la coolbar emploie les couleurs de premier et d'arrière-plan `Foreground` et `Background`. Si la valeur est `False`, les bandes s'affichent selon leurs couleurs par défaut.

En résumé

Ce chapitre vous a expliqué comment on programme les feuilles, à l'aide de quels événements, propriétés et méthodes. En tant qu'arrière-plan de l'application, la feuille est un élément central de la programmation. Les feuilles multiples peuvent être référencées de plusieurs façons. Mais les collections `Forms` que nous avons découvertes aujourd'hui, représentent sans doute la technique la plus simple et la plus fiable.

Vous pouvez paramétrer les diverses propriétés de la feuille, mais également créer vos propres propriétés de feuille. Une fois la nouvelle propriété ajoutée, le code peut définir et lire ses valeurs comme il le ferait pour une propriété normale.

La programmation multidocument (MDI) est un peu laborieuse, mais elle permet d'obtenir des applications complexes, dans lesquelles chaque fenêtre a son propre jeu de données. Vous pouvez commencer la création de vos programmes MDI par l'assistant Création d'applications, puis compléter les détails.

Les contrôles Toolbar et Coolbar enrichissent vos applications. La barre d'outils permet à l'utilisateur d'accéder aux commandes courantes en cliquant simplement sur des boutons. Pour que ces boutons affichent des icônes, vous devez spécifier les fichiers graphiques correspondants dans la liste du contrôle ImageList. Les coolbars sont des barres d'outils coulissantes qui permettent à l'utilisateur de faire place à d'autres boutons ou éléments. Les barres d'outils et coolbars peuvent être proposées sous formes d'options du menu Affichage, afin que l'utilisateur choisisse ce qui lui convient.

Le chapitre suivant explique comment utiliser le disque dur comme source de données externe pour les applications Visual Basic.

Questions-réponses

Q Pourquoi dois-je m'embarrasser du contrôle ImageList pour pouvoir afficher des icônes sur une barre d'outils ou une coolbar ?

R Les contrôles Toolbar et Coolbar ont été conçus pour fonctionner avec le contrôle ImageList. C'est aussi simple que ça. Les développeurs Microsoft auraient pu vous permettre d'affecter directement les icônes dans la liste des propriétés de la barre d'outils ou de la coolbar. Il n'en est pas ainsi. Les icônes affichées correspondent à des éléments indexés dans la liste d'images. Mais cela a un avantage : vous pouvez définir sur une feuille plusieurs contrôles ImageList et les employer comme . Par exemple, vous pouvez utiliser une liste d'images pour les boutons de la barre d'outils lorsque ces boutons sont actifs et disponibles, et une autre liste d'images

lorsque ces boutons sont inactifs (parce que l'utilisateur effectue une tâche telle que les boutons de la barre d'outils ne doivent pas être disponibles). Il suffit pour cela de changer le nom de la liste d'images dans la propriété `ImageList` de la barre d'outils ou de la coolbar ; Visual Basic basculera automatiquement vers l'autre jeu d'icône.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre. Mais également des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Comment l'événement `Resize` permet-il d'assurer le centrage des contrôles sur la feuille ?
2. A quoi sert la propriété `ScaleMode` ?
3. Quelle est la valeur du premier indice lorsqu'on travaille sur une collection d'objets prédéfinie ?
4. Une application SDI ne peut contenir plusieurs feuilles. Vrai ou faux ?
5. Quelle est la différence entre une application SDI et une application MDI ?
6. Où affiche-t-on généralement les barres d'outils sur une feuille ?
7. Quel contrôle contiennent les icônes de la barre d'outils ?
8. Pourquoi l'emplacement du contrôle ImageList sur une coolbar n'a-t-il aucune importance ?
9. Quelle est la différence entre `Spc()` et `Tab()` ?
10. Comment insérer une ligne vierge avec `Print` ?

Exercices

1. Ajoutez à votre Boîte à outils les contrôles Toolbar, Coolbar et ImageList.
2. Décrivez la sortie des instructions suivantes :

```
Form1.Print "Ligne 1";
Form1.Print "Ligne 2"
```

3. Lancez l'assistant Création d'applications pour créer une application de style Explorateur, puis exécutez le programme généré.
4. Ecrivez un code qui affiche sur une feuille les nombres 1 à 100. Séparez chaque nombre d'un espace. N'utilisez aucun contrôle pour la sortie. Incluez le code dans une procédure événementielle `Click` qui déclenchera la sortie.
5. Ecrivez une procédure qui calcule le nombre total de contrôles contenus dans toutes les feuilles d'une application, puis affiche le résultat.

Chapitre 12

Gestion des fichiers

Ce chapitre dévoile le principe des entrées/sorties fichiers (E/S). Le programmeur Visual Basic est amené à travailler sur différents types d'E/S. Les types que nous allons étudier aujourd'hui sont essentiels à la compréhension des autres techniques de traitement de fichiers. Lorsque vous maîtriserez les E/S fichiers, vous serez en mesure d'exploiter les outils plus avancés de gestion de fichiers et de base de données.

Il existe trois types de fichiers : *séquentiels*, *aléatoires* et *binaires*. L'accès séquentiel est le plus simple, mais il a quelques inconvénients : les fichiers séquentiels sont faciles à créer et à lire, mais lents et peu souples. L'accès aléatoire est bien plus rapide et bien plus commode, mais exige aussi des programmes plus complexes que l'accès séquentiel. Quant aux fichiers binaires, ils sont une forme spéciale, compressée, des fichiers d'accès aléatoire.

Les fichiers d'accès aléatoire acceptent tous les types de données que vous puissiez déclarer. Vous apprendrez un peu plus bas à déclarer vos propres types de données, ainsi qu'à utiliser les instructions `Put #` qui permettent de lire ou d'écrire dans les fichiers de données.

Voici ce que nous découvrirons aujourd'hui :

- les types de fichiers ;
- la différence entre les fichiers séquentiels et les fichiers aléatoires ;
- l'importance des numéros de fichiers ;
- comment ouvrir un fichier ;
- comment localiser les numéros de fichier disponibles ;
- les commandes `Print #`, `Write #`, `Read #`, `Get #` et `Put #` ;
- les contrôles de traitement de fichiers.

Les traitements de fichiers

Qui dit programme dit fichiers. Quand bien même un programme n'exploiterait aucun fichier, il n'en résiderait pas moins, lui-même, dans un ou plusieurs fichiers. Qu'il s'agisse d'enregistrer des informations dans une base de données ou de simplement stocker des informations pour leur propre usage (telles que la position d'une fenêtre ou les couleurs préférées de l'utilisateur), la plupart des programmes reposent sur des fichiers.

Beaucoup de commandes Visual Basic sont communes à toutes les formes d'E/S fichiers. Ces commandes ouvrent et ferment des fichiers, spécifient des modes d'accès aux fichiers, recherchent des numéros de fichier disponibles, etc. Cette section explique le rôle et le fonctionnement des traitements de fichiers. Nous commencerons cet exposé par l'instruction qui commence tout programme de traitement de fichiers : `Open`.

L'instruction *Open*

Les fichiers séquentiels et les fichiers aléatoires ont certaines choses en commun. Notamment, l'instruction `Open` permet d'ouvrir l'un comme l'autre type de fichiers. Le type d'accès-fichier opéré par `Open` dépend de l'argument spécifié dans l'instruction. `Open` réserve un gestionnaire de fichier, ou *canal*, pour lire ou écrire dans un fichier. L'instruction `Open` associe au canal un numéro. Dès que ce canal est ouvert, le flux de données peut couler. Du mode selon lequel vous ouvrez le numéro de fichier — lecture, écriture ou les deux — dépendent les modalités du flux de données entre le fichier et l'ordinateur.

Un gestionnaire de fichier, ou canal, est un chemin unique vers un fichier, identifié par le numéro assigné par l'instruction `Open`. A partir du moment où `Open` associe au canal un numéro de fichier, le reste du programme accède au fichier sous ce seul numéro. Une fois qu'il a exécuté l'instruction `Open`, le programme ne se réfère plus au nom du fichier.

Voici le format de l'instruction `Open` :

```
Open "strNomFichier" [For Mode] [AccessRestriction] [LockType] As
[#]intFileNum [Len = intRecordLength]
```

Insistons sur ce point : une fois qu'un numéro d'identification est attribué au fichier de données, le programme n'appelle plus jamais, pour ainsi dire, le fichier par son nom.

Deux arguments sont obligatoires dans l'instruction `Open` : le nom du fichier et son numéro. Les autres arguments sont optionnels. Voici une instruction `Open` "toute nue", incluant les seuls arguments obligatoires :

```
Open "fichier.txt" As #1
```

Cette instruction ouvre `fichier.txt` comme fichier numéro `1`. C'est ce numéro qui servira de référence pour toute entrée ou sortie. Le fichier est ici ouvert en accès aléatoire — `Random` est le mode par défaut quand l'argument *`Mode`* n'est pas spécifié. Toutes les commandes que nous étudierons aujourd'hui s'appliquent aux fichiers texte. Aussi rencontrerez-vous surtout les extensions les plus courantes pour de tels fichiers : .txt et .dat.

Les modes d'accès aux fichiers

Notez que l'argument `Mode` de l'instruction `Open` ne comporte aucun préfixe qui indique son type de données. `Mode` doit être un mot clé spécial qui spécifie le mode d'accès au fichier. Le Tableau 12.1 détaille les valeurs que peut prendre l'argument `Mode`.

Tableau 12.1 : Valeurs possibles de l'argument Mode

Mode	*Description*
`Append`	Ouvre un fichier pour sortie séquentielle, en commençant par la fin du fichier. Si le fichier n'existe pas, Visual Basic le crée. `Append` n'écrase jamais les fichiers de données existants.
`Binary`	Ouvre un fichier en accès binaire. Dans le mode `Binary`, le fichier est accessible au niveau octet, c'est-à-dire que l'on peut y lire ou y écrire octet par octet.
`Input`	Ouvre un fichier pour lecture séquentielle, en commençant par le début du fichier. Les données sont lues dans l'ordre selon lequel elles ont été envoyées au fichier.
`Output`	Ouvre un fichier pour sortie séquentielle, en commençant par le début du fichier. Si le fichier n'existe pas au moment de l'exécution de l'instruction `Open`, Visual Basic le crée. Si le fichier existe, il est écrasé.
`Random`	Ouvre un fichier pour lecture et écriture aléatoires. Dans ce mode, les données peuvent être lues et écrites dans l'enregistrement selon n'importe quel ordre.

L'argument `For Mode`, nous l'avons dit, n'est pas obligatoire dans l'instruction `Open`. Si vous ne spécifiez pas de mode, Visual Basic opère automatiquement en mode `Random`, et va même jusqu'à insérer `For Random` pour vous. Les instructions suivantes illustrent les différents modes d'accès aux fichiers :

```
Open "Quelconque.txt" For Input As #1

Open "Append.txt" For Append As #1

Open "Output.txt" For Output As #1

Open "Random.txt" For Random As #1
```

Cette dernière instruction est équivalente à la suivante :

```
Open "Random.txt" As #1
```

Il convient de mettre en place une logique de gestion d'erreurs, à l'aide de l'instruction `On Error Goto` étudiée au Chapitre 9. Chaque fois que l'on accède à un fichier, une erreur est susceptible de se produire. Une bonne stratégie de gestion d'erreurs vous permettra, à défaut de les résoudre, de contourner élégamment ces problèmes et d'épargner à vos utilisateurs les cascades de messages d'erreur inopportuns.

Les restrictions d'accès

L'argument optionnel `AccessRestriction` permet de restreindre l'exécution de l'instruction `Open` aux modes d'accès `Read`, `Write` ou `Read Write`. De telles restrictions sont notamment appliquées aux fichiers appelés à circuler sur un réseau.

Dans l'accès en lecture seule, `Read`, l'utilisateur peut lire le contenu du fichier, mais pas le modifier. L'accès `Write` permet à l'utilisateur de modifier le fichier, et l'accès `Read Write` lui permet de faire les deux.

Verrouillage des fichiers

L'argument `LockType` spécifie les opérations que d'autres processus peuvent effectuer sur le fichier. Cet argument est notamment d'une grande utilité dans les applications réseau. Il permet de restreindre l'accès au fichier de sorte qu'un seul utilisateur à la fois puisse y accéder ou y écrire. On évite ainsi que deux utilisateurs modifient en même temps un même fichier (ce qui entraînerait inévitablement la perte des modifications de l'un deux utilisateurs).

`LockType` peut prendre les valeurs `Shared`, `Lock Read`, `Lock Write` et `Lock Read Write`. `Shared` permet à tous les utilisateurs d'accéder simultanément au fichier. `Lock Read` verrouille le fichier de sorte que seul l'utilisateur qui est en train de le lire puisse y accéder. `Lock Write` fait de même pour l'utilisateur qui est en train d'écrire dans le fichier. `Lock Read Write` verrouille le fichier pour tous les utilisateurs autres que celui qui lit ou écrit dans le fichier.

La longueur d'enregistrement

La longueur spécifiée par l'argument `Len = intRecordLength` définit, pour les fichiers d'accès aléatoire, la taille des *enregistrements* de données qu'ils recevront du programme. Cette taille doit être spécifiée pour tout accès aux enregistrements d'un fichier. Le premier enregistrement d'un fichier commence à la position `1`, et tous les enregistrements subséquents seront écrits à des positions incrémentées de `1`. La postion dans le fichier d'un enregistrement donné est *N* × `intRecordLength`, où *N* est le numéro de l'enregistrement.

Un enregistrement est une ligne logique qui, dans un fichier, contient un jeu de données complet. Dans un fichier d'inventaire, par exemple, chaque caractéristique d'un élément (descriptif, prix, quantité, etc.) constituerait un enregistrement.

On accède aux enregistrements un peu comme on accède aux éléments d'un tableau de variables. De même que le premier élément d'un tableau est déclaré comme `Array(0)`, le premier élément d'un fichier est stocké à l'enregistrement numéro `1`. Pour tirer le meilleur parti de cette symétrie entre les indices de tableaux et les numéros d'enregistrements, spécifiez `Option Base 1` dans la section de déclarations, ou indexez vos éléments de tableaux à partir de `1`.

Localiser un numéro de fichier disponible

Visual Basic tolère l'ouverture simultanée de plusieurs fichiers, à condition que l'on attribue à chacun un numéro différent. Vous devez être en mesure de déterminer le prochain numéro disponible, notamment si des fichiers sont ouverts dans une fonction qui n'a aucun moyen de savoir si d'autres fonctions ont ouvert des fichiers. Il existe, à cet effet, la fonction `FreeFile()` qui renvoie le prochain numéro disponible. Cette fonction garantit que le numéro utilisé dans une instruction n'est pas déjà attribué. Voici le format de `FreeFile()` :

```
FreeFile[(intRangeNumber)]
```

L'argument *intRangeNumber*, optionnel, spécifie la plage dans laquelle le numéro renvoyé doit être compris : `1-255` ou `256-511`. En l'absence d'argument `intRangeNumber`, la plage par défaut est `1-255`. Cette valeur est, dans presque tous les cas, conservée telle quelle, car il serait surprenant qu'un programme ait à ouvrir plus de *256* fichiers. Si l'argument *intRangeNumber* n'est pas spécifié, les parenthèses de la fonction sont inutiles.

Les instructions suivantes utilisent `FreeFile()` pour obtenir un numéro et ouvrir un fichier sans ce numéro :

```
intFileNumber = FreeFile
Open "AccPay.Dat" For Output As intFileNumber
```

Ainsi, `FreeFile()` permet de s'assurer qu'un numéro de fichier n'est pas déjà attribué. Cette attention peut sembler inutile pour une petite application qui ne traite que quelques fichiers ; pourtant, même dans des cas pareils, `FreeFile()` aide à prévenir les erreurs.

Il vaut mieux éviter d'inclure `FreeFile` dans une instruction `Open`, comme ici :

```
Open strFileName For Output As FreeFile()
```

Ça marche, mais vous n'avez, vous, aucun moyen de connaître le numéro de fichier, qui peut être indispensable pour des opérations ultérieures.

L'instruction *Close*

Tous les fichiers ouverts avec `Open` doivent, tôt ou tard, être refermés. C'est le rôle de l'instruction `Close`. Cette instruction n'accepte pour seuls paramètres que le numéro des fichiers à fermer. Voici le format complet de `Close` :

```
Close # intFileNumber[, intFileNumber2][,...intFileNumberX]
```

On peut spécifier, dans une même instruction `Close`, autant de numéros que nécessaire. Si aucun numéro n'est spécifié, tous les fichiers ouverts sont fermés. C'est, par exemple, un bon moyen de refermer proprement l'application.

Le code du Listing 12.1 ouvre deux fichiers séquentiels — l'un en lecture, l'autre en écriture — en utilisant le prochain numéro disponible, puis les referme.

Listing 12.1 : `FreeFile()` permet d'obtenir un numéro de fichier libre

```
1:  Dim intReadFile As Integer, intWriteFile As Integer
2:  ' Gère le fichier d'entrée.
3:  intReadFile = FreeFile   ' Obtient le numéro du premier fichier.
```

```
4:  Open "AccPay.Dat" For Input As intReadFile
5:  ' Gère le fichier de sortie.
6:  intWriteFile = FreeFile   ' Obtient le numéro du fichier suivant.
7:  Open "AccPayOut.Dat" For Output As intWriteFile
8:  '
9:  ' Ici, le code chargé d'envoyer au fichier
10: ' de sortie le contenu du fichier d'entrée
11: ' (voir plus loin).
12: Close intReadFile
13: Close intWriteFile
```

Dans ce code, on n'a pas à préciser de numéros de fichiers, car `FreeFile()` (lignes 3 et 6) renvoie les numéros disponibles qui sont stockés comme entiers nommés.

En ne refermant pas tous les fichiers ouverts, vous vous exposez au risque — certes limité par les performances des matériels modernes — d'une perte de données. Si la machine est brutalement mise hors tension, les fichiers encore ouverts peuvent être endommagés. Vous devez donc avoir soin de refermer un fichier dont vous n'avez plus besoin ; dans tous les cas, le système s'en chargera à la fermeture de l'application.

Une même instruction `Close` peut, comme on l'a vu, refermer autant de fichiers que l'on veut. Cette simple ligne ferme tous les fichiers ouverts :

```
Close
```

Les deux lignes suivantes, en revanche, ne referment que deux fichiers précis :

```
Close 3
Close 6
```

En spécifiant dans `Close` les numéros de fichiers, vous ne refermez que les fichiers indésirables, mais pouvez continuer à travailler sur les autres.

Les fichiers séquentiels

Vous connaissez maintenant les instructions qui permettent d'ouvrir et de fermer les fichiers et de spécifier le mode d'accès. Cette section illustre de plusieurs exemples les opérations d'E/S en accès séquentiel. Vous découvrirez notamment que l'instruction `Print` dont nous avons vu, au chapitre précédent, qu'elle permettait d'envoyer du texte à une feuille, peut aussi envoyer du texte à un fichier.

En créant un fichier de type séquentiel, vous spécifiez que l'accès à ce fichier devra se faire séquentiellement, c'est-à-dire que l'application devra y lire et y écrire en allant du début à la fin. Cette obligation de lire et d'écrire séquentiellement est la plus grande faiblesse des fichiers séquentiels.

Un fichier séquentiel est un fichier dans lequel on lit et écrit selon l'ordre allant du début à la fin.

Pour exploiter le fichier, le programme devra le prendre en entier, du début à la fin. Même pour ne mettre à jour qu'un seul octet sur les 1000 octets d'un fichier donné, il faudra traiter les 999 autres octets.

Le type séquentiel est généralement employé pour les fichiers texte (tels que des fichiers de paramètres), ou pour stocker de petites quantités de données n'exigeant pas une haute vitesse d'accès. Cette section présente les fonctions qui permettent de gérer les fichiers séquentiels.

L'instruction *Print #*

Pour qu'un programme puisse utiliser un fichier, il doit d'abord l'ouvrir. Il faut ensuite lui envoyer des données. `Print #` est l'instruction la plus couramment employée à cet effet. `Print #` ne peut écrire que dans les fichiers d'accès séquentiel. En voici le format :

```
Print #intFileNumber, [OutputList]
```

Ici, *`intFileNumber`* est le numéro du fichier ouvert dans lequel on veut écrire. *`OutputList`* peut avoir les valeurs suivantes :

```
[Spc(intN1) | Tab[(intN2)]] [Expression] [charPos]
```

`Spc()` et `Tab()` fonctionnent avec `Print #` de la même façon qu'elles fonctionnent avec l'instruction `Print` étudiée au chapitre précédent.

`Spc()` ou `Tab()`, il faut choisir. Vous ne pouvez utiliser les deux en même temps. Le Tableau 12.2 détaille les composants de *`OutputList`*.

`Tab()` peut être incluse dans `charPos`. Elle a alors le même rôle qu'en début d'instruction `Print #`. Si vous omettez `charPos`, le prochain caractère apparaîtra sur la ligne suivante du fichier.

La procédure du Listing 12.2 ouvre un fichier nommé `Print.txt`, y écrit les chiffres 1 à 5, puis referme proprement le fichier.

Tableau 12.2 : Le contenu de `Print #` décrit la sortie de la méthode

Composant	*Description*
`Spc(intN1)`	Insère des espaces dans la sortie, *intN1* étant le nombre d'espaces à insérer.
`Tab(intN2)`	Positionne le point d'insertion au numéro de colonne absolu *intN2*. Sans arguments, `Tab` positionne le point d'insertion au début de la prochaine *zone d'écriture* (il y a une *zone d'écriture* toutes les 14 espaces).
`Expression`	Expression chaîne ou numérique qui contient les données à envoyer au fichier.
`CharPos`	Spécifie l'emplacement du point d'insertion pour le prochain caractère. Un point-virgule (;) indique que le prochain caractère doit suivre immédiatement le précédent.

Listing 12.2 : Ecriture dans un fichier séquentiel avec `Print #`

```
1:  Private Sub cmdFile_Click()
2:    Dim intCtr As Integer   ' Compteur de boucle.
3:    Dim intFNum As Integer  ' Numéro de fichier.
4:    Dim intMsg As Integer   ' Valeur de renvoi de MsgBox().
5:    intFNum = FreeFile
6:    ' Vous pouvez changer le chemin.
7:    Open "C:\Print.txt" For Output As #intFNum
8:
9:    ' Décrit la procédure.
10:   intMsg = MsgBox("Fichier Print.txt ouvert !")
11:
12:   For intCtr = 1 To 5
13:     Print # intFNum, intCtr    ' Envoie le compteur de boucle.
14:     intMsg = MsgBox("Ecriture du chiffre " & intCtr & " dans Print.txt")
15:   Next intCtr
16:
17:   Close # intFNum
18:
19:   intMsg = MsgBox("Fichier Print.txt fermé !")
20: End Sub
```

Si vous exécutez ce code, vous verrez s'afficher plusieurs boîtes de message indiquant la progression de la procédure. La procédure indique qu'elle a ouvert le fichier (ligne 10), puis elle écrit dans le fichier (ligne 13), puis montre le résultat (ligne 14). Enfin, elle ferme le fichier (ligne 17) et vous en fait part, avec un contentement non dissimulé (ligne 19).

Pour vérifier la sortie, ouvrez le fichier `Print.txt` dans le Bloc-notes. Voici ce que vous devriez voir :

```
1
2
3
4
5
```

Le Listing 12.2 met en œuvre une instruction `Print #` simple (line 13). Aucun argument n'ayant spécifié la position du point d'insertion, la procédure écrit chaque chiffre sur une ligne différente.

Créer et écrire dans un fichier, ça ne servirait pas à grand-chose si l'on ne pouvait obtenir des informations en retour. C'est ce que nous ferons dans la section suivante.

L'instruction *Input #*

Vous écriviez ? Eh bien lisez maintenant. L'instruction `Input #` permet de lire les données séquentielles. Les données séquentielles doivent être lues séquentiellement, c'est-à-dire dans l'ordre et le format dans lesquels elles ont été écrites. Voici le format de `Input #` :

```
Input # intFileNumber, Variable1[, Variable2][, ...VariableN]
```

`Input #` requiert un numéro de fichier ouvert, ainsi qu'une variable qui stocke les données lues. L'instruction `Input #` doit respecter le format de l'instruction `Print #` qui a servi à écrire dans le fichier. Si les données écrites par `Print #` étaient délimitées par des virgules, ces virgules doivent se retrouver dans `Input #`.

Pour qu'une série de variables écrites sur une seule ligne puissent être correctement lues par `Input #`, il faut soit l'écrire avec `Write` plutôt que `Print #`, soit insérer manuellement des virgules délimitatrices. Dans une variable numérique, `Input #` suspend sa lecture à chaque espace, virgule ou caractère de fin de ligne. Dans une chaîne, `Input #` suspend sa lecture à chaque virgule ou caractère de fin de ligne, sauf si la chaîne contient des guillemets.

L'instruction suivante lit, dans un fichier séquentiel ouvert, cinq variables placées sur une même ligne :

```
Input #intFileNumber V1, V2, V3, V4, V5
```

Pour pouvoir lire les données, `Input #` doit reprendre le format de l'instruction `Print #` qui a servi à écrire dans le fichier.

Le fonctionnement de `Input #` est assez simple : l'opération qu'elle effectue est l'exact reflet de l'opération `Print #` correspondante. La prochaine section présente l'instruction `Write #`, qui permet d'écrire des fichiers de données dans un format plus général que `Print #`, de sorte que l'on n'a pas à se soucier de la cohérence absolue de `Input #` avec l'instruction de sortie initiale.

L'instruction *Write #*

`Write #` est donc une autre commande d'écriture séquentielle. Elle ne diffère de `Print #` que sur quelques points. D'abord, toutes les données écrites par `Write #` sont délimitées par des virgules. Ensuite, `Write #` place automatiquement les données `String` entre guillemets (les guillemets apparaissent dans le fichier), place automatiquement les données `Date` entre caractères dièse (`#`), écrit les données `Boolean` sous la forme `#TRUE#` et `#FALSE#`, et envoie les données `Null` et les codes d'erreurs sous la forme `#NULL#` et `#Error errorcode#`, respectivement. (*`errorcode`* est un numéro identifiant l'erreur de sortie, par exemple un lecteur non trouvé. Vous trouverez dans l'aide en ligne de Visual Basic une liste des codes d'erreur et leur signification.)

La délimitation par virgules est parfois nécessaire pour que les données puissent être lues par certains tableurs et logiciels de publipostage. Ce format a aussi l'avantage de limiter les risques d'erreurs à la lecture, l'instruction `Input #` n'étant pas tenue de refléter exactement l'instruction `Write #` correspondante.

Pour une lecture plus simple et plus fiable des données, utilisez `Write #` plutôt que `Print #`.

Voici le format de `Write #` :

```
Write # intFileNumber, [OutputList]
```

Ici, *`OutputList`* est une liste des variables contenues dans le fichier qui doivent être lues en particulier.

Nous avons expliqué, plus haut, comment l'instruction `Print #` pouvait écrire une valeur par ligne. `Print #` accepte également les mêmes options de formatage que l'instruction `Print` simple.

Par exemple, pour écrire des valeurs l'une après l'autre sur une même ligne, il suffit de placer un point-virgule après la variable `intCtr` (voir Listing 12.3).

Listing 12.3 : Le point-virgule permet d'écrire plusieurs valeurs sur une même ligne

```
1:  Private Sub cmdFile_Click()
2:    Dim intCtr As Integer   ' Compteur de boucle.
3:    Dim intFNum As Integer  ' Numéro de fichier.
4:    Dim intMsg As Integer   ' Valeur de renvoi de MsgBox().
5:    intFNum = FreeFile
6:    ' Vous pouvez changer le chemin.
7:    Open "C:\Print.txt" For Output As #intFNum
8:
9:    ' Décrit la procédure.
10:   intMsg = MsgBox("Fichier Print.txt ouvert")
11:
12:   For intCtr = 1 To 5
13:   Print # intFNum, intCtr;    ' Remarquez le point-virgule.
14:   intMsg = MsgBox("Ecriture du chiffre " & intCtr & " dans Print.txt")
15:   Next intCtr
16:
17: Close # intFNum
18:
19:   intMsg = MsgBox("Fichier Print.txt fermé")
20: End Sub
```

Si vous exécutez la procédure, voici ce qui s'affichera dans le fichier créé :

```
1 2 3 4 5
```

Les chiffres sont sur la même ligne, séparés par des espaces. `Print #` insère ces espaces à la place du signe plus imaginaire qui précède les nombres positifs.

N'hésitez pas à expérimenter les différents paramètres de `Print #` afin d'en bien saisir le fonctionnement.

Une fois les données écrites dans un fichier, la lecture de ces données a souvent lieu dans une autre procédure, voire une autre application. Les procédures du Listing 12.4 écrivent les données dans un fichier, puis lisent ces mêmes données au travers de variables.

Listing 12.4 : Ecriture et lecture dans un fichier au sein d'une même procédure

```
1:  Private Sub cmdFileOut_Click ()
2:    ' Crée le fichier séquentiel.
3:    Dim intCtr As Integer   ' Compteur de boucle.
4:    Dim intFNum As Integer  ' Numéro de fichier.
5:    intFNum = FreeFile
6:
7:    Open "Print.txt" For Output As #intFNum
8:
```

```
9:    For intCtr = 1 To 5
10:     Print # intFNum, intCtr;    ' Ecrit le compteur de boucle.
11:   Next intCtr
12:
13:   Close # intFNum
14: End Sub
15:
16: Private Sub cmdFileIn_Click ()
17:   ' Lit le fichier séquentiel.
18:   Dim intCtr As Integer   ' Compteur de boucle.
19:   Dim intVal As Integer   ' Valeur lue.
20:   Dim intFNum As Integer  ' Numéro de fichier.
21:   Dim intMsg As Integer   ' Valeur de renvoi de MsgBox().
22:   intFNum = FreeFile
23:   Open "Print.txt" For Input As #intFNum
24:
25:   For intCtr = 1 To 5
26:     Input # intFNum, intVal
27:     ' Affiche les résultats dans la fenêtre Exécution.
28:     intMsg = MsgBox("Lecture du chiffre " & intVal & " dans Print.txt")
29:   Next intCtr
30:
31:   Close # intFNum
32:   intMsg = MsgBox("Le fichier Print.txt file est maintenant fermé")
33: End Sub
```

La première procédure du Listing 12.4 écrit les données dans le fichier, données qui sont ensuite lues par la procédure `cmdFileIn_Click()`.

Examinez maintenant la procédure du Listing 12.5 qui crée un fichier nommé `Write.txt`.

Listing 12.5 : Ecriture dans un fichier séquentiel avec Write

```
1:  Private cmdFile_Click ()
2:    Dim intCtr As Integer   ' Compteur de boucle.
3:    Dim intFNum As Integer  ' Numéro de fichier.
4:    intFNum = FreeFile
5:
6:    Open "c:\Write.txt" For Output As #intFNum
7:
8:    For intCtr = 1 To 5
9:      Write # intFNum, intCtr;    ' Ecrit le compteur de boucle.
10:   Next intCtr
11:
12:   Close # intFNum
13: End Sub
```

Exécutez cette procédure, puis ouvrez dans le Bloc-notes le fichier créé. Vous constaterez immédiatement la différence entre `Print #` et `Write #`. Voici le contenu de `Write.txt` :

```
1,2,3,4,5,
```

Si l'on n'avait pas inclus le point-virgule de formatage, chaque chiffre serait apparu sur sa propre ligne, et il n'y aurait pas eu de virgules. (Dans un tel cas, `Write #` et `Print #` se comportent de la même manière.)

Plus vous travaillerez sur des fichiers séquentiels, plus vous découvrirez de "trucs" propres à améliorer le code. Il peut être utile, par exemple, d'indiquer en guise de premier fragment de données le nombre total de valeurs que contient le fichier. Les programmes qui liront ces données, pourront ainsi exploiter cette information dans des boucles, etc.

Après tout ce que vous venez d'ingurgiter, cela va sans doute vous surprendre : les programmeurs Visual Basic n'exploitent que très rarement les fichiers séquentiels. Cela ne signifie pas pour autant que vous avez perdu votre temps, car les principes étudiés dans le cadre des données séquentielles s'appliquent en fait à tous les autres types d'accès. Ouf !

Les fichiers aléatoires

Si les fichiers séquentiels doivent toujours être lus et écrits dans le même ordre, les *fichiers d'accès aléatoire* (ou *fichiers aléatoires* tout court) peuvent être lus et écrits dans n'importe quel ordre. Ainsi, les enregistrements écrits dans un fichier client de type aléatoire pourront, par la suite, être lus dans l'ordre que l'on veut. Si le fichier client était séquentiel, il faudrait parcourir tous les enregistrements précédents avant de pouvoir consulter un enregistrement particulier.

Un fichier d'accès aléatoire, ou fichier aléatoire, est un fichier dans lequel on peut lire et écrire dans n'importe quel ordre.

Comme l'accès séquentiel, l'accès aléatoire n'est aujourd'hui plus guère utilisé dans sa forme "pure". Les contrôles d'accès aux données et procédures de traitement de fichiers plus avancés lui font une rude concurrence. Toutefois, les techniques d'accès aux fichiers de base de données reposent, en grande partie, sur les principes que vous allez maintenant étudier.

Nous en profiterons également pour aborder une nouvelle technique de programmation : les *types de données personnalisés*. Les fichiers aléatoires contiennent généralement des enregistrements de données ; Visual Basic vous permet de définir des type de données en parfaite adéquation avec les enregistrements lus et écrits dans les fichiers séquentiels.

L'accès aléatoire

Le traitement des fichiers aléatoires ressemble, en beaucoup de points, au traitement de fichiers séquentiels. Les instructions `Open` et `Close`, par exemple, fonctionnent de la même manière. Seul change le mode d'accès.

Si vous ne spécifiez pas le mode d'accès au fichier, Visual Basic utilise par défaut le mode aléatoire, et insère lui-même l'argument `For Random`*. Si, par exemple, vous tapez :*

```
Open "Random.txt" As #1
```

Visual Basic changera automatiquement la ligne en :

```
Open "Random.txt" For Random As #1
```

L'instruction suivante ouvre un fichier en accès aléatoire :

```
Open "Random.txt" For Random As #1
```

Rien n'empêche d'ouvrir un fichier en accès aléatoire pour l'exploiter en mode séquentiel. Simplement, on perd les avantages offerts par le mode aléatoire. Mais cela est parfois utile, notamment dans le cas d'enregistrements stockés dans un ordre donné que l'on voudrait imprimer ou afficher dans cet ordre précis.

Voici un exemple qui illustre bien la différence entre accès séquentiel et accès aléatoire. Supposons un fichier contenant dix lignes d'inventaire. Pour lire la sixième ligne (ou enregistrement), il faudrait, en mode séquentiel, lire d'abord les cinq premiers enregistrements, puis, après le sixième, lire encore les quatre derniers enregistrements. En mode aléatoire, ont irait directement lire le sixième enregistrement, pour ensuite fermer le fichier.

Il en va de même pour l'écriture. Si, en mode séquentiel, on voulait modifier le huitième enregistrement du même fichier d'inventaire, il faudrait lire les dix enregistrements, modifier le huitième, puis réécrire les dix enregistrements dans le fichier. En mode aléatoire, au contraire, on interviendrait directement sur le huitième enregistrement.

Evidemment, pour un fichier de dix enregistrements, les avantages de l'accès aléatoire ne sont pas si importants. Mais dans le cas d'un fichier contenant 10 000 enregistrements, quel gain de temps ! Et quelle économie de ressources système !

Les instructions *Get* et *Put*

Les instructions `Put #` et `Get #` sont à l'accès aléatoire ce que les instructions `Print #` et `Input #` sont à l'accès séquentiel. La différence principal est que `Print #` et `Input #` traitent un élément de données à la fois, jusqu'à la fin du fichier. Ces instructions ne peuvent en aucun cas accéder directement à un enregistrement spécifique pour le mettre à jour.

En outre, le format de `Put #` et `Get #` diffère légèrement de celui de `Print #` et `Input #` :

```
Put [#]intFileNum, [intRecNum,] Variable
Get [#]intFileNum, [intRecNum,] Variable
```

Ces instructions utilisent un numéro d'enregistrement (*intRecNum*). C'est ce numéro qui permet de lire ou d'écrire directement un enregistrement précis. La numérotation des enregistrements commence à `1`. Les variables lues ou écrites peuvent prendre n'importe quel type de données — même `Array` (tableau), et même un type par vous défini (voir section suivante). Cette capacité de traiter comme unité des variables de tous types est pour beaucoup dans la puissance de l'accès aléatoire.

Les exemples de la section suivante incluent des procédures qui lisent et écrivent des enregistrements spécifiques dans un fichier aléatoire.

Les types de données personnalisés

Nous vous avons déjà entretenu des variables et des tableaux de variables. Vous allez maintenant apprendre à créer vos propres types de données, en combinant des types de données existants. Ces types de données personnalisés sont parfois appelés *structures* ou — gare à la confusion — *enregistrements*.

Supposons que vous vouliez créer un programme pour gérer votre carnet d'adresses. Vous pourriez déclarer des variables individuelles pour chaque champ : une variable pour le nom de famille, une autre pour le prénom, etc. Cela fonctionnerait certainement. En revanche, pour peu que la liste des contacts soit importante, la programmation impliquée deviendrait considérable.

Il serait bien plus simple de définir un type de données pour chaque catégorie d'information, et que vous manipuleriez comme n'importe quelle variable individuelle.

Un type de données personnalisé consiste en fait de types existants, regroupés pour former un nouveau type. Un tel regroupement est appelé *déclaration composite*.

Un type de données personnalisé est composé de types existants — éventuellement même d'autres types personnalisés.

C'est à l'aide de l'instruction `Type` que vous créez vos propres types de données. En voici le format :

```
[Private | Public] Type TypeName
  VarName1 [(ArraySize)) As ExistingType [*StringLength]
  VarName2 [(ArraySize)] As ExistingType [*StringLength]
     :
     :
End Type
```

Le nom du type à créer (*`TypeName`*) suit le mot clé `Type`. Ce nom peut être tout sauf un mot réservé, un mot clé ou un nom de variable déclarée. Si, par exemple, on a déjà déclaré une variable nommée `Client`, le nouveau type de données ne pourra s'appeler `Client`.

Tous les nouveaux types doivent être déclarés au niveau module. On ne peut les déclarer à l'intérieur d'une procédure. Le type peut aussi être déclaré dans le module de feuille, mais alors comme `Private`, c'est-à-dire qu'il ne sera reconnu que dans ce module de feuille particulier.

Examinez le code du Listing 12.6.

Listing 12.6 : L'instruction Type permet de déclarer les nouveaux types de données

```
1:  ' Page module du projet.
2:  Type UserType
3:    strFName As String
4:    strLName As String
5:  End Type
6:  Public Names As UserType
```

Ce code crée un nouveau type nommé `UserType`. Ce nouveau type contient deux chaînes, `strFName` et `strLName`. La ligne 6 déclare la variable `Names` comme type `UserType`.

Ici, `UserType` n'est pas une variable, mais bien le type que l'on a défini. Le nom de la variable est `Names`, et `strFName` et `strLName` sont des membres (ou champs) de cette variable. Le nouveau type est intégré au langage Visual Basic pour le temps de l'exécution du programme. On peut maintenant déclarer des variables `Integer`, `Boolean` ou... `UserType`.

Pour accéder aux champs individuels qui constituent le type de données, on indique le nom de la variable, suivi d'un point, suivi du nom du champ.

Les instructions suivantes initialisent et traitent la variable que nous venons de déclarer :

```
Names.strFName = "Serge"
Names.strLName = "Pichot"
lblFName.Caption = "Prénom : " & Names.strFName
lblLName.Caption = "Nom : " & Names.strLName
```

Pour limiter la taille des variables chaînes utilisées dans une structure, on inclut dans la déclaration l'option `* StringLength` (où `StringLength` est la longueur), juste après `As String`. La chaîne de longueur fixe définit `StringLength` comme longueur absolue de la chaîne. Cela est nécessaire lorsque les structures doivent être lues ou écrites aléatoirement dans un fichier. La longueur fixe garantit que tous les enregistrements écrits auront la même taille, afin que l'accès aléatoire se passe sans problème.

Le Listing 12.7 reprend le code du Listing 12.6, mais en imposant aux chaînes une longueur fixe.

Listing 12.7 : Les chaînes de longueur fixe permettent de spécifier la longueur des enregistrements

```
1:  ' Page module du projet.
2:  Type UserType2
3:    strFName As String * 8
4:    strLName As String * 20
5:  End Type
6:  Public Names As UserType2
```

Les chaînes de longueur fixe définissent pour les chaînes une taille maximale. Notez que le contenu de la chaîne peut ne pas occuper entièrement l'espace réservé, auquel cas Visual Basic comble le vide avec des espaces. Ainsi, toutes les variables déclarées comme `UserType2` et écrites en accès aléatoire auront la même longueur d'enregistrement, quelle que soit la taille réelle des données contenues dans la variable.

Les procédures du Listing 12.8 mettent en œuvre les principes de l'accès aléatoire.

Listing 12.8 : Ecriture dans un enregistrement particulier

```
1:  Private Sub cmdCreate_Click()
2:    ' Cette procédure crée le fichier.
3:    Dim intFile As Integer   ' Numéro de fichier disponible.
4:    Dim intCtr As Integer    ' Compteur de boucle.
5:
```

```
 6:    intFile = FreeFile
 7:    Open "c:\Random.Txt" For Random As #intFile Len = 5
 8:
 9:    ' La boucle parcourt les numéros et écrit dans le fichier.
10:   For intCtr = 1 To 5
11:     Put # intFile, intCtr, intCtr  ' Le numéro d'enregistrement correspond
        ➥aux données.
12:   Next intCtr
13:
14:   Close intFile
15: End Sub
16:
17: Private Sub cmdChange_Click()
18:   ' Cette procédure modifie l'enregistrement n°3.
19:   Dim intFile As Integer   ' Numéro de fichier disponible.
20:
21:   intFile = FreeFile
22:   Open "c:\Random.Txt" For Random As #intFile Len = 5
23:
24:   ' Ecrit un nouvel enregistrement n°3.
25:   Put #intFile, 3, 9   ' Value = 9.
26: Close # intFile
27: End Sub
28:
29: Private Sub cmdDisplay_Click()
30:   ' Cette procédure affiche le fichier
31:   Dim intFile As Integer   ' Numéro de fichier disponible.
32:   Dim intVal As Integer    ' Valeur lue.
33:   Dim intCtr As Integer    ' Compteur de boucle.
34:   Dim intMsg As Integer    ' Valeur de renvoi de MsgBox().
35:   intFile = FreeFile
36:   Open "c:\Random.Txt" For Random As #intFile Len = 5
37:
38:   intMsg = MsgBox("Fichier Random.Txt ouvert...")
39:
40:   ' La boucle parcourt les enregistrements et les affiche.
41:   For intCtr = 1 To 5
42:     Get # intFile, intCtr, intVal
43:     intMsg = MsgBox("Lecture de " & intVal & " dans Random.Txt")
44:   Next intCtr
45:   Close # intFile
46:
47:   intMsg = MsgBox("Fichier Random.Txt fermé")
48: End Sub
```

Notez que l'instruction aléatoire `Open` de la ligne 7 utilise l'option `Len`. A la ligne 11, l'instruction `Put #` écrit un enregistrement aléatoire avec une longueur de `5` ; c'est l'option `Len` qui spécifie cette longueur. Ce paramètre est capital : si la longueur d'enregistrement n'était pas spécifiée, `Put #` et `Get #` ne sauraient jamais jusqu'où chercher

dans le fichier un enregistrement particulier. (La formule pour trouver un enregistrement est *`RecordNumber * RecordLength`*.)

La feuille de l'application du Listing 12.8 contient trois boutons. Le premier crée le fichier, le deuxième affiche le fichier, le troisième modifie le fichier. Chacun des boutons déclenche une procédure événementielle. Créez cette petite application et exécutez-la : cliquez sur le bouton `cmdCreate`, puis sur le bouton `cmdDisplay`. La boîte de message affiche les données écrites. Cliquez sur le bouton `cmdChange`, puis de nouveau sur le bouton `cmdDisplay`. Une fois le fichier modifié, l'enregistrement n° 3 contient la valeur `9` (au lieu de `3`). La sous-routine qui a provoqué ce changement, `cmdChange_Click()`, a simplement écrit la valeur `9` dans l'enregistrement n° 3, à l'aide de l'instruction `Put #`.

Imbrication de types de données personnalisés

Vous avez créé votre premier type de données personnalisé. Mais peut-on inclure un type personnalisé à l'intérieur d'un autre type personnalisé ? Parfaitement. L'un des champs, au lieu d'être un type interne à Visual Basic, est un type que vous avez créé vous-même. Prenez simplement garde de bien déclarer le type à inclure *avant* de déclarer le type qui doit le contenir.

Le Listing 12.9 donne l'exemple d'un type personnalisé nommé `Address`, imbriqué comme champ dans un autre type personnalisé.

Listing 12.9 : Les types personnalisés peuvent être inclus dans un autre type personnalisé

```
1:  ' Dans la section de déclarations du module de code.
2:  Type Address
3:    strStreet As String
4:    strCity As String
5:    strZip As String
6:  End Type
7:
8:  Type UserType3
9:    strFName As String * 10
10:   strLName As String * 25
11:   typAddr As Address  ' Autre type de données.
12: End Type
13:
14: Public Names As UserType3  ' Déclare une variable
    ➥d'application.
```

Le code du Listing 12.10 initialise ces champs et montre comment on accède aux champs dans les champs.

Listing 12.10 : Le type de données public peut servir dans tous les modules

```
1:  Names.strFName = "Serge"
2:  Names.strLName = "Pichot"
3:
4:  Names.typAddr.strStreet = "Clergeot"
5:  Names.typAddr.strCity = "Perrochonville"
6:  Names.typAddr.strZip "99000"
7:
8:  ' Traite les données.
9:  lblFName.Caption = "Prénom : " & Names.strFName
10: lblLName.Caption = "Nom : " & Names.strLName
11: lblAddr.Caption = "rue : " & Names.strAddr.strStreet
12: lblCty.Caption = "Ville : " & Names.strAddr.strCity
13: lblZip.Caption = "Code postal : " & Names.strAddr.strZip
```

Les contrôles de fichiers

Parmi les boîtes de dialogue communes présentées au Chapitre 9, les boîtes de dialogue Ouvrir et Enregistrer permettent à l'utilisateur de manipuler des fichiers. Il peut y sélectionner des dossiers, des lecteurs et même des lecteurs réseau.

Pour les traitements de fichiers impliqués par vos applications, il peut être nécessaire de spécifier un répertoire (ou *dossier* depuis Windows 95), un lecteur ou un nom de fichier, sans passer par des boîtes de dialogue.

Visual Basic dispose à cet effet de trois types spéciaux de zones de liste :

- **Zone de liste Dossier.** Permet à l'utilisateur de sélectionner un dossier.
- **Zone de liste Lecteur.** Permet à l'utilisateur de sélectionner un lecteur de disque.
- **Zone de liste Fichier.** Permet à l'utilisateur de sélectionner un nom de fichier.

La Figure 12.1 montre une feuille contenant ces trois zones de liste spéciales.

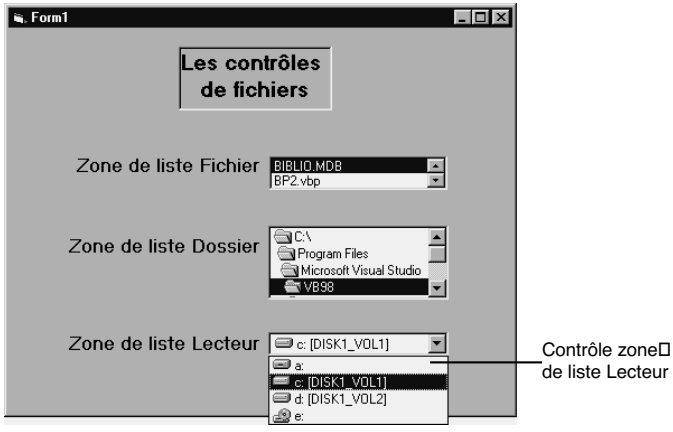
Ces contrôles de listes spéciaux ne fonctionneront pas conjointement, à moins que vous ne les programmiez à cet effet. Par exemple, si les trois contrôles apparaissent sur la feuille, et que vous exécutiez l'application, un changement dans la liste Lecteur ne modifiera pas le dossier ou le nom de fichier affiché dans les deux autres listes. Pour que les choses se passent ainsi, vous devez écrire des procédures événementielles propres à synchroniser les contrôles entre eux.

Figure 12.1
Les trois types de zones de liste servant à la manipulation des fichiers.

La zone de liste Lecteur

La zone de liste Lecteur permet à l'utilisateur de sélectionner un lecteur de disque. Ce contrôle ausculte lui-même le système à la recherche des divers types de lecteurs : locaux, distants, disque dur, disquette, CD-ROM, etc. Lorsque l'utilisateur clique sur la liste, les différents choix possibles sont représentés par des icônes (voir Figure 12.2).

Figure 12.2
L'utilisateur fait son choix dans la liste Lecteur.



La zone de liste Lecteur affiche d'abord le lecteur depuis lequel l'application a été lancée. Vous pouvez modifier ce comportement par défaut en spécifiant, dans le code, un autre lecteur.

La zone de liste Dossier

La zone de liste Dossier permet à l'utilisateur de sélectionner un dossier. Ce contrôle analyse le système pour établir l'arborescence des dossiers existants. Lorsque l'utilisateur clique sur la liste, les différents choix possibles sont représentés par des icônes et organisés hiérarchiquement, selon le standard Windows.

Rappelez-vous que la zone de liste Dossier ne sait pas quel lecteur est sélectionné dans la liste Lecteur. C'est à vous de relier les deux contrôles, comme nous l'expliquons à la fin de ce chapitre.

La zone de liste Dossier affiche d'abord le dossier contenant l'application. Vous pouvez modifier ce comportement par défaut en spécifiant, dans le code, un autre dossier.

La zone de liste Fichier

La zone de liste Fichier permet à l'utilisateur de sélectionner un fichier. Ce contrôle analyse le système pour établir la liste de tous les fichiers existants. Lorsque l'utilisateur clique sur la liste, les différents choix possibles sont représentés par des icônes et organisés hiérarchiquement, selon le standard Windows.

Pas plus que la zone de liste Dossier, la zone de liste Fichier ne sait pas quel lecteur (ou quel dossier) est sélectionné dans la liste Lecteur (ou dans la liste Dossier). C'est à vous de relier les contrôles dans le code, comme nous l'expliquons à la fin de ce chapitre.

La zone de liste Fichier affiche d'abord la liste des fichiers contenus dans le dossier de l'application. Vous pouvez modifier ce comportement par défaut en spécifiant, dans le code, un autre dossier pour la liste Dossier, puis en liant à cette liste la liste Fichier.

Les commandes de traitement de fichiers

Visual Basic supporte plusieurs commandes liées aux lecteurs, dossiers et fichiers (voir Tableau 12.3).

`RmDir` générera une erreur si le dossier à supprimer contient encore des fichiers.

Tableau 12.3 : Commandes liées aux lecteurs et dossiers

Commande	*Description*
`ChDrive strDrive`	Spécifie à la place du lecteur par défaut le lecteur *strDrive*.
`ChDir strDirectory`	Spécifie à la place du dossier par défaut le dossier *strDirectory*. Si aucun lecteur n'est spécifié dans la chaîne, Visual Basic opte pour le lecteur courant.
`Kill strFileSpec`	Supprime les fichiers (représentés par des jokers) spécifiés dans *strFileSpec*.
`MkDir strDirectory`	Crée le dossier spécifié dans *strDirectory*.
`RmDir strDirectory`	Supprime le dossier spécifié dans *strDirectory*.

Outre les commandes du Tableau 12.3, Visual Basic supporte la fonction `Dir()` qui vérifie si des fichiers existent bien, et la fonction `CurDir()` qui renvoie le nom du dossier courant.

Supposons que vous vouliez affecter aux zones de liste Lecteur et Dossier le chemin `C:\MesFichiers`. Il suffit d'insérer dans la procédure `Form_Load()` le code suivant :

```
ChDrive "C:"
ChDir "\MesFichiers"
```

En s'affichant sur la feuille, les zones de liste Lecteur, Dossier et Fichier pointeront toutes vers `C:\MesFichiers`, plutôt que sur le dossier courant de l'application.

La fonction `Dir()` requiert un peu plus d'explications. Imaginons que vous vouliez vérifier qu'un fichier nommé VENTES98.DAT existe bien à la racine du lecteur D. Vous pouvez procéder ainsi :

```
If (Dir("c:\VENTES98.DAT")) = "VENTES98.DAT" Then
  intMsg = MsgBox ("The file exists")
Else
  intMsg = MsgBox ("The file does not exist")
End If
```

La fonction `Dir()` renvoie le nom de fichier passé comme argument. Le nom de fichier est renvoyé seulement si le fichier correspondant réside dans le dossier spécifié dans l'argument. Si `Dir()` ne renvoie pas le nom de fichier, c'est que le fichier n'existe pas sur le lecteur désigné.

`Dir()` accepte également les jokers :

```
Dir("c:\Ventes*.DAT")
```

`Dir()` renvoie alors le nom du premier fichier correspondant aux jokers passés — si pareil fichier existe. Une fois passée la première spécification joker, vous pouvez appeler de nouveau `Dir()` sans parenthèses ni arguments. Visual Basic renverra les noms de fichiers répondant aux jokers, jusqu'à ce que le dernier fichier soit trouvé. Si `Dir` renvoie une chaîne `Null` (`""`), vous devez inclure une spécification de fichier dans le prochain appel de `Dir()` ; autrement, `Dir` générerait une erreur.

Pour affecter à la zone de liste Lecteur un lecteur spécifique, vous devez définir la propriété `Drive` du contrôle comme ceci :

```
drvDisk.Drive = "d:\"
```

La lettre de lecteur `D:` s'affiche alors en haut de la liste. Si l'utilisateur sélectionne un autre lecteur, l'événement `Change()` de la zone de liste Lecteur se produit. Pour définir comme lecteur par défaut le lecteur spécifié par l'utilisateur, il faut inclure dans `drvDisk_Change()` l'instruction suivante :

```
ChDrive drvDisk.Drive
```

Pour que le lecteur s'affiche dans la zone de liste Dossier, définissez la propriété `Drive` du contrôle à l'aide l'instruction suivante :

```
dirDirect.Drive = drvDisk.Drive
```

Cette instruction d'affectation définit, dans la liste Dossier, le lecteur spécifié par l'utilisateur. Elle peut être ajoutée à la procédure événementielle `drvDisk_Change()`.

Lorsque l'utilisateur change de dossier dans la zone de liste Dossier, l'événement `Change` du contrôle se produit. Dans la procédure événementielle `Change`, vous pouvez définir le dossier spécifié par l'utilisateur comme dossier courant à l'aide de l'instruction suivante :

```
ChDir dirDirect.Path
```

La zone de liste Dossier supporte une méthode d'accès un peu particulière : il suffit de la propriété `ListIndex`. La valeur de `ListIndex` est `-1` pour le dossier sélectionné, `-2` pour le dossier parent du dossier sélectionné, `-3` pour le parent du parent, et ainsi de suite. La valeur de `ListIndex` est `0` pour le premier sous-dossier du dossier sélectionné, `1` pour le sous-dossier suivant, et ainsi de suite.

Pour n'afficher que certains fichiers dans la zone de liste Fichier, affectez à la propriété `Pattern` du contrôle une chaîne spécifiant un filtre :

```
filFiles.Pattern = "*.vbp; *.frm"
```

Vous pouvez inclure autant de spécifications que vous le souhaitez, sous la forme de jokers inclus entre les guillemets de la chaîne. La zone de liste Fichier se met immédiatement à jour selon les filtres spécifiés. Lorsque l'utilisateur sélectionne un fichier, l'événement `Change` de la zone de liste Fichier se produit, et le fichier sélectionné est affecté à la propriété `FileName`. Comme pour la zone de liste Lecteur, le fichier sélectionné apparaît également avec une valeur `ListIndex` de `-1`.

Si l'utilisateur a sélectionné un chemin d'accès, vous pouvez répercuter ce choix dans la zone de liste Fichier à l'aide de l'instruction d'affectation suivante :

```
filFiles.Path = dirDirect.Path
```

En résumé

Ce chapitre vous a appris à manipuler les fichiers au niveau le plus fondamental. Maintenant que vous maîtrisez ce bases, vous êtes prêt à affronter les contrôles et commandes de traitement de fichiers plus avancés, tels que les fonctionnalités base de données de Visual Basic.

Vous avez également appris à lire et écrire dans des fichiers séquentiels et aléatoires. Ces fichiers sont utiles pour stocker des valeurs textuelles. Une fois que l'on connaît les commandes, la programmation de ces fichiers est relativement simple. Ainsi, l'instruction `Write #` reflète l'instruction `Read #`, et l'instruction `Get #` reflète l'instruction `Put #`, etc.

Les contrôles de gestion de fichiers de la Boîte à outils, une fois installés et correctement paramétrés dans votre application, permettent à l'utilisateur de parcourir ses lecteurs, dossiers et fichiers. Ces contrôles forment un complément intéressant aux boîtes de dialogue communes Ouvrir et Enregistre, et permettent d'obtenir des informations spécifiques sur le contenu du disque.

Le chapitre suivant vous enseigne à envoyer des données vers un autre type de périphérique : l'imprimante.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Combien de fichiers peut-on fermer avec une même instruction `Close` ?
2. Quelle fonction renvoie le prochain numéro de fichier disponible ?
3. Que se passe-t-il si vous ouvrez un fichier en accès séquentiel `Output` et que ce fichier existe déjà ?
4. Que se passe-t-il si vous ouvrez un fichier en accès séquentiel `Append` et que ce fichier existe déjà ?
5. Quel type de fichier l'instruction suivante ouvre-t-elle ?

```
Open "Test.dat" For Append As #1
```

6. Pourquoi les instructions aléatoires `Open` doivent-elles connaître la longueur d'enregistrement de leurs données ?
7. Pourquoi faut-il spécifier la longueur absolue des chaînes dans un type de données personnalisé quand ces chaînes doivent être lues et écrites dans un fichier aléatoire ?
8. Quelle instruction permet de définir un nouveau type de données ?
9. Le code suivant déclare une nouvelle variable nommée `CustRec`. Vrai ou faux ?

```
Type CustRec
   strFName As String * 10
   strLName As String * 15
   curBalance As Currency
  blnDiscount As Boolean
End Type
```

10. Quelle est la différence entre la fonction `Dir()` avec arguments et la fonction `Dir()` sans arguments ?

Exercices

1. **Chasse au bogue :** Mauricette a des problèmes avec ses fichiers. Lorsqu'elle essaie de lancer une application contenant l'instruction ci-dessous, elle obtient une erreur. Pouvez-vous lui expliquer pourquoi son instruction ne marche pas ? (On suppose qu'un dossier Factures existe bien à la racine du lecteur C.)

```
RmDir "C:\Factures"
```

2. Ecrivez une procédure qui crée un fichier séquentiel contenant les informations suivantes : nom, âge et couleur préférée. Ecrivez dans ce fichier cinq enregistrements (chaque enregistrement devant contenir un nom, un âge et une couleur). Utilisez une seule boucle `For` pour écrire ces informations dans le fichier. Conseil : initialisez trois tableaux, un pour chaque catégorie de données.
3. Créez de toutes pièces une boîte de dialogue semblable à la boîte de dialogue commune Ouvrir. Utilisez pour ce faire les zones de liste Lecteur, Dossier et Fichier, ainsi que des boutons de commande OK et Annuler. Faites en sorte que les listes Dossier et Fichier se modifient chaque fois que l'utilisateur sélectionne un lecteur ou un dossier différent. (Naturellement, vous vous servirez toujours de la boîte de dialogue commune Ouvrir. Il s'agit d'un exercice illustrant les relations entre ces divers contrôles.)

PB6

Lire et écrire des fichiers

Ce Projet bonus devrait vous permettre d'approfondir votre compréhension de l'accès séquentiel. L'application que nous vous invitons à créer intègre également le contrôle Common Dialog, sous la forme d'une boîte de dialogue commune Ouvrir. Mais vous allez surtout mettre en œuvre les procédures de gestion de fichiers étudiées au Chapitre 12. Dans cette application, il s'agit de visualiser le contenu des fichiers. Voici les tâches à accomplir :

- permettre à l'utilisateur de sélectionner un fichier dans la boîte de dialogue Ouvrir ;
- permettre à l'utilisateur de modifier la couleur d'arrière-plan du fichier à l'aide d'une boîte de dialogue Couleur ;
- permettre à l'utilisateur de redimensionner la feuille, et faire en sorte que le programme ajuste automatiquement la disposition des contrôles ;
- intégrer au code de la boîte de dialogue Ouvrir un gestionnaire d'erreurs, de sorte que l'utilisateur puisse, en cliquant sur Annuler, maintenir à l'écran le fichier courant sans faire d'autre sélection ;
- permettre à l'utilisateur de visualiser les fichiers .BAT et .TXT, tout en limitant à 4096 octets la taille des fichiers lisibles dans l'application.

La Figure PB6.1 montre le visualiseur après qu'un fichier a été sélectionné.

Vous ajouterez d'abord les éléments graphiques de la feuille, puis vous écrirez le code chargé de répondre aux actions de l'utilisateur.

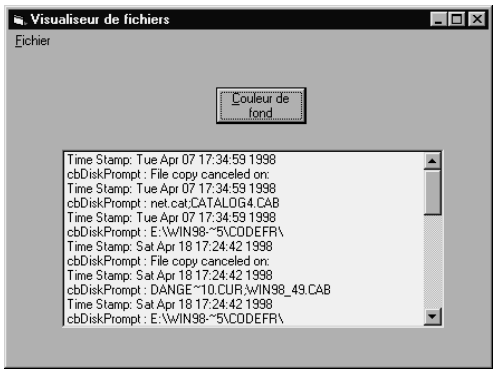


Figure PB6.1
L'utilisateur peut visualiser les fichiers .BAT et .TXT dans la zone de liste.

Créer l'interface

Ajoutez à la feuille les contrôles détaillés dans le Tableau PB6.1.

Dans le Créateur de menus, vous devez indenter tous les éléments sauf le premier, Fichier, qui constitue le nom du menu. Les éléments indentés apparaîtront comme options de ce menu Fichier.

Tableau PB6.1 : Contrôles et propriétés de la feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmFile`
Feuille	`Caption`	`Visualiseur de fichiers`
Feuille	`Height`	`4620`
Feuille	`Width`	`6570`
Zone de liste	`Name`	`lstFile`
Zone de liste	`Height`	`2205`
Zone de liste	`Left`	`720`
Zone de liste	`Top`	`1320`
Zone de liste	`Width`	`4815`

Tableau PB6.1 : Contrôles et propriétés de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Bouton de commande	`Name`	`cmdColor`
Bouton de commande	`Caption`	`&Couleur de fond`
Bouton de commande	`Height`	`495`
Bouton de commande	`Left`	`2760`
Bouton de commande	`Top`	`480`
Bouton de commande	`Width`	`1215`
Elément de menu 1	`Caption`	`&Fichier`
Elément de menu 1	`Name`	`mnuFile`
Elément de menu 2	`Caption`	`&Ouvrir`
Elément de menu 2	`Name`	`mnuFileOpen`
Elément de menu 3	`Caption`	`-`
Elément de menu 3	`Name`	`mnuFileSep1`
Elément de menu 4	`Caption`	`&Quitter`
Elément de menu 4	`Name`	`mnuFileExit`
Common Dialog	`Name`	`comFile`
Common Dialog	`DialogTitle`	`Ouvrir un fichier`
Common Dialog	`InitDir`	`c:\`
Common Dialog	`Filter`	`Texte (*.txt)¦*.txt¦Batch (*.bat)¦*.bat`
Common Dialog	`CancelError`	`True`
Common Dialog	`MaxFileSize`	`4096`

Pour paramétrer le contrôle Common Dialog, double-cliquez sur l'entrée `(Personnalisé)` de la fenêtre Propriétés. Les Pages de propriétés (voir Figure PB6.2) offrent une méthode plus simple que la saisie des valeurs dans la fenêtre Propriétés.

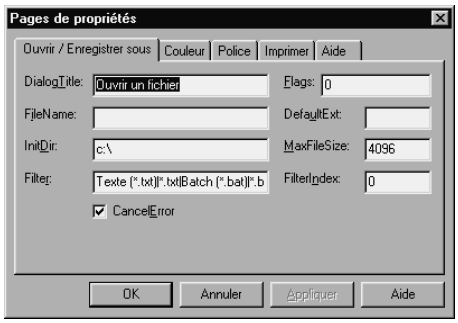


Figure PB6.2
Utilisez la boîte de dialogue personnalisée pour entrer les propriétés du contrôle CommonDialog.

Vous n'aurez pas à définir de propriétés liées à la couleur. La boîte de dialogue Couleur ne nécessite pas de tels réglages avant l'exécution du programme. Les sélections de couleur de l'utilisateur n'affecteront pas les propriétés liés au fichier définies pour le contrôle Common Dialog.

Ajouter le code

Le Listing PB6.1 fournit le code événementiel du projet.

Cliquez sur le menu Outils, puis choisissez l'option Ajouter une procédure. La boîte de dialogue reproduite en Figure PB6.3 s'affiche. Tapez le nom de la procédure, sélectionnez son type et sa portée, et Visual Basic insère automatiquement les lignes d'emballage. Deux lignes de moins à taper, c'est toujours ça !

Figure PB6.3
Laissez Visual Basic insérer la première et la dernière lignes de procédure.

Listing PB6.1 : Chargement du fichier dans un contrôle zone de liste

```
1:  Private Sub cmdColor_Click()
2:     ' Pour changer le contrôle d'arrière-plan
3:     ' de la zone de liste, l'utilisateur
4:     ' se servira de la boîte de dialogue Couleur.
5:     comFile.ShowColor
6:     lstFile.BackColor = comFile.Color
7:  End Sub
8:
9:  Private Sub Form_Resize()
10:   Dim intMsg As Integer   ' Valeur de renvoi de MsgBox().
11:   ' Si l'utilisateur redimensionne la feuille,
12:   ' ajuste la taille de la zone de liste.
13:   '
14:   ' Cette procédure événementielle s'exécute
15:   ' au premier chargement de la feuille.
16:   '
17:   ' S'assure que la feuille est assez grande
18:   ' pour afficher la zone de liste.
19:     If (frmFile.Width < 400) Or (frmFile.Height < 3500) Then
20:      ' Masque la zone de liste et
21:      ' avertit l'utilisateur.
22:       lstFile.Visible = False
23:       intMsg = MsgBox("La feuille est trop petite pour afficher le
          ➥fichier", _ vbCritical)
24:     Else
25:       ' Active l'affichage de la zone de liste,
26:       ' au cas où.
27:       lstFile.Visible = True
28:       ' Ajuste la taille de la zone de liste.
29:       ' Ajuste la position du bouton de commande.
30:       lstFile.Width = frmFile.Width - 1440
31:       lstFile.Height = frmFile.Height - 2500
32:       cmdColor.Left = (frmFile.Width / 2) - 500
33:     End If
34: End Sub
35:
36: Private Sub mnuFileExit_Click()
37:   ' Option de fermeture du programme.
38:   End
39: End Sub
40:
41: Private Sub mnuFileOpen_Click()
42:    Dim strFileLine As String
43:    ' Gère le bouton Annuler.
44:    On Error GoTo comErrorHandler
45:    '
46:    ' Affiche la boîte de dialogue Ouvrir.
47:    comFile.ShowOpen
48:    ' Continue si l'utilisateur clique sur OK,
49:    ' passe au gestionnaire d'erreurs s'il clique sur Annuler.
```

Listing PB6.1 : Chargement du fichier dans un contrôle zone de liste *(suite)*

```
50:     '
51:     ' Ouvre le fichier sélectionné par l'utilisateur.
52:     Open comFile.FileName For Input As #1
53:     ' Vide la zone de liste pour faire de la place.
54:       lstFile.Clear
55:     '
56:     ' Lit une ligne complète du fichier.
57:     Line Input #1, strFileLine
58:     lstFile.AddItem strFileLine
59:     '
60:     ' Poursuit la lecture et remplit la zone de liste
61:     ' jusqu'à ce que la fin du fichier soit atteinte.
62:     Do Until (EOF(1))
63:       Line Input #1, strFileLine
64:       lstFile.AddItem strFileLine
65:     Loop
66:     ' Ferme le fichier.
67:     Close
68: comErrorHandler:
69:     ' Ne fait rien si l'utilisateur clique sur Annuler.
70: End Sub
```

Analyse

La première procédure événementielle, lignes 1 à 7, affiche le contrôle Common Dialog à l'aide de la méthode `ShowColor`. Lorsque l'utilisateur fait une sélection, la ligne 6 ajuste en conséquence l'arrière-plan de la zone de liste et le fichier est affiché sur un fond de la couleur choisie. (La couleur s'applique à l'arrière-plan de la zone de liste même si aucun fichier n'est visualisé.) Cette couleur restera en arrière-plan tant que l'utilisateur n'en sélectionnera pas une autre.

A la ligne 9 commence la deuxième procédure événementielle. Si l'utilisateur redimensionne la feuille, la procédure événementielle `Form_Resize()` s'exécute automatiquement. (Elle s'exécute également au premier chargement de la feuille.)

Cette procédure d'ajustement fait en sorte que le bouton de commande et la zone de liste restent centrés, quelle que soit la taille de la feuille. Si l'utilisateur "ratatine" la feuille au point que la zone de liste ne puisse plus s'afficher, les deux événements suivants ont lieu (ligne 19) :

- La zone de liste est masquée. (Sinon, Visual Basic générerait une erreur, car il ne peut faire ce qui lui est demandé sur une feuille si petite.)
- L'utilisateur est averti par une boîte de message que la feuille est trop petite.

En supposant que la zone de liste n'a pas été masquée, la ligne 27 en ajuste la taille selon les nouvelles propriétés `Width` et `Height` de la feuille. La position du bouton de commande est également modifiée en fonction de cette même propriété `Width`.

A la ligne 41 commence la procédure événementielle la plus longue du programme. Cette procédure ouvre et lit le fichier sélectionné. Lorsque l'utilisateur clique sur Fichier, Ouvrir, la boîte de dialogue commune Ouvrir un fichier s'affiche (voir Figure PB6.4). La procédure événementielle appplique pour cela la méthode `ShowOpen` au contrôle Common Dialog.

Figure PB6.4
L'utilisateur peut choisir un fichier .BAT ou .TXT dans la boîte de dialogue Ouvrir un fichier.

Evidemment, la liste des fichiers affichée dans la boîte de dialogue Ouvrir dépend du contenu du disque de l'utilisateur, et ne sera donc pas identique à la Figure PB6.4.

Dans la boîte de dialogue Types de fichier, l'utilisateur peut sélectionner les types .TXT et .BAT (tous les dossiers sont affichés, quelle que soit la sélection). Il peut aller chercher son fichier dans n'importe quel dossier de son PC, voire d'un autre PC du réseau.

La ligne 44 définit le gestionnaire d'erreur chargé de répondre au bouton Annuler. Si l'utilisateur clique sur Annuler, le code saute à la fin de la procédure événementielle, et laisse tel quel tout le contenu de la feuille.

Quand l'exécution du programme a atteint la ligne 52, l'utilisateur a sélectionné un fichier. La ligne 52 ouvre ce fichier en accès séquentiel, en mode lecture seule.

Le fichier est ouvert en lecture seule afin que le programme ne puisse en modifier le contenu.

La ligne 54 vide la zone de liste afin de faire place au contenu du fichier nouvellement sélectionné. Les lignes 57 et 58 lisent la première ligne du fichier et ajoutent cette ligne à la zone de liste à l'aide de la méthode `AddItem`. La ligne 57 introduit une nouvelle commande : `Line Input #`. Vous connaissez déjà `Input #` et `Get #`. `Line Input #` ne devrait donc pas vous poser de problème. En voici le format :

```
Line Input #intFileNumber, strLineVar
```

`Line Input #` lit une ligne complète (ou enregistrement) depuis le fichier, et la stocke dans la variable chaîne *`strLineVar`*. Si c'est `Input #` qui avait été employée, la ligne n'aurait peut-être pas été lue en entier. En effet, les lignes contiennnent sans doute des virgules et des espaces, et vous savez que `Input #` arrête sa lecture chaque fois qu'un tel caractère est rencontré — même si c'est au milieu de la ligne.

Une fois la première ligne lue et envoyée à la zone de liste, la ligne 62 amorce une boucle qui lira le reste du fichier. La fonction `EOF()` permet de localiser la fin d'un fichier. Le fichier peut contenir plusieurs lignes, et le programme ne peut en connaître à l'avance le nombre exact. `EOF()` renvoie la valeur `True` lorsque la fin du fichier est atteinte. Si `True` est renvoyée, la boucle cesse de lire le fichier et le programme poursuit son exécution à la ligne 66.

La ligne 62 aurait pu être celle-ci :

```
Do Until (EOF(1) = True)
```

Certes, la line 62 aurait pu interroger la valeur renvoyée par `EOF()`. Mais cette façon de faire aurait, en vérité, donné un programme moins efficace. La fonction `EOF()` renvoie une valeur `True` ou `False` selon que la condition de fin de fichier est satisfaite ou non. Les fonctions, vous vous en souvenez sans doute, deviennent leur valeur de renvoi. En l'occurrence, l'appel de `EOF()` devient lui-même `True` ou `False`, et l'instruction conditionnelle `Do Until` est donc tout à fait inutile.

La ligne 67 ferme le fichier, et le programme se poursuit pour permettre à l'utilisateur d'en choisir un nouveau.

Chapitre 13

Gestion de l'imprimante

Vous apprendrez, dans ce chapitre, comment on envoie une sortie vers l'imprimante. Vous maîtrisez déjà l'environnement de développement, connaissez l'essentiel du langage et êtes en mesure de créer des applications fonctionnelles. Il s'agit maintenant de permettre à ces applications d'envoyer des données à l'imprimante.

Sachez que l'interface d'impression Visual Basic est un peu plus complexe que la plupart des composants étudiés jusqu'ici. Imprimer des données depuis un programme Visual Basic est en général une opération délicate. Les sections suivantes décrivent les divers outils disponibles à cet effet.

Voici ce que nous découvrirons aujourd'hui :

- les objets `Printer` ;
- comment déterminer les paramètres d'impression de l'utilisateur ;
- l'instruction `Is TypeOf` ;
- les méthodes d'impression ;
- la similitude entre les méthodes d'impression et les méthodes de feuilles ;
- les techniques d'impression ;
- comment avertir l'utilisateur ;
- comment imprimer des feuilles.

La collection d'objets *Printers*

Supposons que vous vouliez imprimer une copie du contenu de la fenêtre Code. Pour ce faire, vous choisissez Fichier, Imprimer. Un PC peut disposer de plusieurs imprimantes. Vos applications doivent avoir accès à toutes les imprimantes connectées, ainsi qu'aux éventuels fax internes qui fonctionnent comme des imprimantes. Il suffit, pour envoyer l'impression, de sélectionner le périphérique adéquat comme imprimante par défaut de Visual Basic. Visual Basic transmettra toutes les sorties à cet imprimante, ignorant la spécification par défaut du système, et cela, jusqu'à ce que l'application se termine ou jusqu'à ce que vous définissiez une autre imprimante par défaut.

Rien de plus simple, donc, que d'imprimer vos listings. Il n'en va pas de même pour imprimer des sorties depuis un programme Visual Basic. Avant d'entrer dans les détails, nous allons présenter un jeu d'objets spécial : la collection `Printers`.

La collection `Printers` est la liste de toutes les imprimantes connectées au PC qui exécute l'application, incluant les modems internes faisant office de fax. Cette collection ne concerne pas le PC sur lequel l'application est développée, mais bien celui sur lequel elle est exécutée.

La collection `Printers` étant la liste des imprimantes du système exploitant l'application, son contenu change d'un système à l'autre. Si, entre deux exécutions, l'utilisateur désinstalle une imprimante, la collection `Printers` en sera modifiée. Comment accéder aux imprimantes de la collection courante ? C'est ce que nous vous invitons à découvrir.

La collection `Printers` est la même liste d'imprimantes qui apparaît dans la boîte de dialogue Imprimantes de Windows.

Accéder à la collection *Printers*

Comme pour les autres types de listes internes que nous avons étudiés jusqu'ici, le programme peut se référer aux objets de la collection `Printers` au moyen de valeurs d'index. La première imprimante (imprimante par défaut) a l'indice `0`, la deuxième l'indice `1`, et ainsi de suite. On peut ainsi déterminer le nombre d'imprimantes du système en incluant dans une boucle `For` la référence `Printers.Count-1`. L'instruction `For Each` permet de parcourir la liste des imprimantes sans qu'il soit besoin de spécifier un numéro (voir Listing 13.1).

L'instruction `Set Printer` permet de définir l'une des imprimantes comme imprimante par défaut. L'instruction suivante définit comme imprimante par défaut de Visual Basic la deuxième imprimante du système :

```
Set Printer = Printers(1)  ' Change l'imprimante par défaut.
```

Il n'est toutefois pas toujours évident de déterminer en cours d'exécution l'imprimante à utiliser. Comment savoir de quel type d'imprimante il s'agit ? On ne peut interroger que des propriétés spécifiques. Si, par exemple, vous devez envoyer la sortie vers une imprimante supportant un format de papier particulier, il faudra boucler à travers toutes les imprimantes du système, jusqu'à ce vous trouviez le périphérique adéquat.

Interroger les propriétés

Le Tableau 13.1 présente les propriétés d'imprimante les plus importantes, qui permettent de chosir une imprimante en fonction des besoins de l'application. Ces propriétés sont, pour la plupart, associées à des constantes nommées. On peut ainsi, pour rechercher notre format de papier, interroger les propriétés à l'aide de constantes nommées telles que `vbPRPSLetter` et `vbPRPSLetterSmall`, plutôt que de simples valeurs comme `1` ou `2`. Les constantes nommées sont un peu plus longues à saisir, mais rendent le programme plus clair et la maintenance plus aisée.

Faites dans l'aide en ligne une recherche sur une propriété pour obtenir la liste des constantes nommées qu'elle supporte.

Tableau 13.1 : Propriétés des imprimantes

Propriété	*Description*
`ColorMode`	Détermine si l'imprimante peut imprimer en couleur.
`Copies`	Spécifie le nombre de copies à imprimer. (Cette propriété peut est définie par l'utilisateur dans la boîte de dialogue Imprimer de votre application, ou par vous-même dans le code.)
`CurrentX`, `CurrentY`	Renvoie ou spécifie les cordonnées X et Y auxquelles le prochain caractère (ou dessin) apparaîtra.
`DeviceName`	Contient le nom de l'imprimante, tel que Epson Stylus Color 500.
`DriverName`	Contient le nom du pilote d'impression. (Plusieurs imprimantes d'un même fabriquant peuvent exploiter le même pilote.)

Tableau 13.1 : Propriétés des imprimantes *(suite)*

Propriété	*Description*
`Duplex`	Détermine si l'imprimante peut imprimer recto-verso.
`Height`	Revoie la hauteur de la page imprimée pour l'imprimante sélectionnée (en mesures `ScaleMode`).
`Orientation`	Renvoie ou définit l'orientation de la page (portrait/paysage).
`Page`	Renvoie le numéro de la page courante.
`PaperBin`	Renvoie ou définit le bac d'alimentation utilisé. (Toutes les imprimantes ne disposent pas de bacs multiples.)
`PaperSize`	Renvoie ou définit le format de papier courant.
`Port`	Renvoie le port de l'imprimante.
`PrintQuality`	Renvoie la résolution de l'imprimante.
`TrackDefault`	Si la valeur est `False`, les paramètres d'impression courants sont conservés lorsqu'on change d'imprimante par défaut ; si la valeur est `True`, ces paramètres sont modifiés à chaque changement d'imprimante par défaut.
`Width`	Renvoie la largeur de la page imprimée pour l'imprimante sélectionnée (en mesures `ScaleMode`).
`Zoom`	Renvoie ou définit le pourcentage d'échelle de l'impression. Pour une valeur de `Zoom` de `75`, par exemple, la page sera imprimée à 75 % de sa taille réelle. (Cette propriété n'est pas supportée par toutes les imprimantes.)

A l'exécution, les propriétés de l'objet `Printer` s'ajustent à celles de l'imprimante par défaut du système. Si une nouvelle imprimante par défaut est sélectionnée, les propriétés se modifient automatiquement. La plupart des propriétés présentées au Tableau 13.1 peuvent être définies à l'exécution.

On se sert également des propriétés du Tableau 13.1 pour appliquer les méthodes spécifiques d'impression décrites à la section suivante.

Le code du Listing 13.1 parcourt les imprimantes courantes du système.

Listing 13.1 : Le code prend connaissance de chaque imprimante du système

```
1:  Dim prnPrntr As Printer
2:  For Each prnPrntr In Printers  ' Boucle dans la collection.
3:    frmMyForm.Print prnPrntr.DeviceName
4:  Next
```

Ce code se contente d'afficher sur la feuille courante le nom de chaque imprimante du système.

A la ligne 1, une variable est déclarée comme de type `Printer`. En progressant dans votre apprentissage, vous découvrirez que les variables peuvent être déclarées pour à peu près n'importe quel type : `Printer`, `Form`, etc. Ici, la variable `prnPrntr` permet à la boucle `For Each` de parcourir les imprimantes du système. L'instruction `For` équivalente serait :

```
For prnPrntr = 1 to (Printers.Count - 1)
```

Rappelez-vous que `Printer` et `Form` sont des objets Visual Basic. Vous en apprendrez plus à propos des objets au Chapitre 16.

Afficher sur la feuille le nom de chaque imprimante, ça ne sert pas à grand-chose. Néanmoins, la boucle mise en œuvre dans cet exemple fournit le modèle de la plupart des traitements d'imprimante.

Pour continuer dans nos exemples, le Listing 13.2 recherche, parmi toutes les imprimantes du système, une imprimante capable d'imprimer un graphique en couleur.

Listing 13.2 : Recherche d'une imprimante couleur sur le système de l'utilisateur

```
1:  Dim prnPrntr As Printer
2:  For Each prnPrntr In Printers
3:    If prnPrntr.ColorMode = vbPRCMColor Then
4:      ' Définit l'imprimante couleur comme imprimante par défaut.
5:      Set Printer = prnPrntr
6:      Exit For  ' Ne cherche pas plus loin.
7:    End If
8:  Next   ' Continue la boucle si nécessaire.
```

Contrôle de la sortie

L'objet `Printer` permet d'envoyer des données vers l'imprimante par défaut sans se soucier du type du périphérique ni du port. Pour envoyer une sortie (documents, images, etc.) vers l'imprimante, on applique à l'objet `Printer` des méthodes. L'objet `Printer` est d'une programmation un peu fastidieuse, mais il permet de développer des procédures généralistes qui simplifieront la gestion des sorties dans les programmes subséquents.

Pour envoyer les données à une imprimante autre que l'imprimante par défaut, il faut d'abord la définir à l'aide de l'instruction `Set Printer`.

Une fois sélectionnée l'imprimante par défaut, on peut envoyer la sortie vers l'imprimante de l'utilisateur. Cette section explique comment contrôler l'objet `Printer` et diriger du texte vers l'imprimante. Vous découvrirez, dans le chapitre suivant, des méthodes et commandes graphiques qui peuvent également s'appliquer à l'objet `Printer`.

Avec l'objet `Printer`, on *construit* la sortie. En d'autres termes, envoyer une sortie à l'objet `Printer` ne signifie pas imprimer réellement. Ce sont les méthodes `NewPage` ou `EndDoc` qui amorcent réellement l'impression. (Si aucune méthode `EndDoc` n'a été appliquée, l'impression se lance automatiquement à la fermeture de l'application.)

Imprimer vers l'objet *Printer*

La méthode `Print` offre le plus simple moyen de diriger une sortie vers l'objet `Printer`. Les instructions suivantes envoient un message vers l'imprimante :

```
Printer.Print "Ce rapport présente les résultats du "
Printer.Print dteStart; " au "; dteFinish; "."
Printer.Print "Pour plus d'informations, contactez M. Pichot."
```

En fait, Windows se charge des détails de l'impression. Ainsi, si l'imprimante n'a plus de papier ou est hors tension lorsque l'impression démarre, Windows envoie à l'utilisateur un message d'avertissement (voir Figure 13.1). Celui-ci peut cliquer sur Réessayer s'il a réglé le problème, ou sur Annuler pour décommander la tâche d'impression.

On peut aussi bien envoyer vers l'imprimante des littéraux, des variables, des valeurs de contrôles, etc. En fait, tout ce que l'on peut envoyer à la feuille avec `Print`, on peut l'envoyer à l'envoyer à l'objet `Printer`. Voici un exemple :

Figure 13.1
Windows avertit l'utilisateur des erreurs d'impression.

```
sngTaxRate = 12.5
strTitle = "Encyclopédie du gibolin"
Printer.Print "Ouvrage acheté : " & strTitle
Printer.Print "Montant de la taxe : " & sngTaxRate
```

Sans arguments, la méthode `Print` imprime des lignes vierges :

```
Printer.Print      ' Imprime une ligne vierge.
```

Il convient de signaler à l'utilisateur le début de l'impression. La dernière section de ce chapitre explique comment envoyer ce type d'avertissements.

Nous avons vu, au Chapitre 11, que la méthode `Print` permettait d'envoyer une sortie vers la feuille. Comme vous le voyez ici, `Print` est une méthode généraliste capable d'envoyer une sortie vers tout objet valide qui accepte les données textuelles.

La méthode `NewPage` permet d'afficher la sortie en haut de la page suivante :

```
Printer.NewPage  ' Passer à la page suivante.
```

L'échelle de la sortie

L'échelle permet, par exemple, de créer des marges qui seront respectées par toutes les méthodes d'impression. Une fois définie la propriété `ScaleMode`, qui spécifie le type de mesures employées par le programme, vous pouvez déterminer la disposition de la sortie sur la page à l'aide des autres propriétés d'échelle. Le Tableau 13.2 présente ces propriétés.

Les méthodes suivantes définissent une marge supérieure de 3 cm et une marge gauche de 5 cm :

```
Printer.ScaleMode = vbCentimeters ' Echelle centimètres.
Printer.ScaleTop = 3
Printer.ScaleLeft = 5
```

Les méthodes `Print` subséquentes respecteront ces spécifications.

Tableau 13.2 : Propriétés d'échelle

Propriété	*Description*
`ScaleLeft`	Définit l'origine de la coordonnée horizontale X pour la zone d'impression. Par exemple, une valeur `ScaleLeft` de `10` déplace la marge de gauche de 10 unités `ScaleMode`.
`ScaleMode`	Spécifie l'unité de mesure utilisée pour l'échelle. Les valeurs les plus couramment employées pour l'impression de texte sont `vbPoints` (valeur `2`), `vbCharacters` (largeur de caractères par défaut de l'imprimante), `vbInches` et `vbCentimeters`.
`ScaleHeight`	Modifie le système de coordonnées verticales de l'objet `Printer`.
`ScaleTop`	Définit l'origine de la coordonnée verticale Y pour la zone d'impression. Par exemple, une valeur `ScaleTop` de `5` déplace la marge supérieure de 5 unités `ScaleMode`.
`ScaleWidth`	Modifie le système de coordonnées horizontales de l'objet `Printer`.

Les propriétés *CurrentX* et *CurrentY*

A moins que les propriétés `ScaleHeight` et `ScaleWidth` ne définissent un autre système de coordonnées `X,Y`, les propriétés `CurrentX` et `CurrentY` de l'objet `Printer` ont toutes deux pour valeur de départ `0` (origine `0, 0`). (Les coordonnées utilisent toujours les unités de mesure spécifiées par `ScaleMode`.) Vous pouvez modifier cette origine pour imprimer une sortie à une position précise sur la page.

Les propriétés `CurrentX` et `CurrentY` respectent toujours les marges définies par `ScaleLeft` et `ScaleTop`. Ainsi, la paire de coordonnées `CurrentX, CurrentY` correspond au premier caractère en haut à gauche de la page, à l'intérieur des marges.

Le code suivant imprime un message à 15 lignes de la marge supérieure et à 25 caractères de la marge gauche :

```
Printer.ScaleMode = VbCharacters
Printer.CurrentY = 14   ' La valeur de départ est 0 !
Printer.CurrentX = 24
Printer.Print "Ce n'est plus Catilina qui est à nos portes, c'est la mort."
```

Plusieurs méthodes, appliquées à l'objet `Printer`, permettent de contrôler le processus d'impression. Ainsi, la méthode `NewPage` déjà mentionnée :

```
Printer.NewPage  ' Passer à la page suivante.
```

La méthode `KillDoc` permet d'annuler l'impression à tout moment de la préparation :

```
Printer.KillDoc  ' Ne pas envoyer la sortie vers l'imprimante.
```

`KillDoc` retire complètement la sortie de l'objet `Printer`. Si l'utilisateur veut imprimer le document par la suite, la sortie devra être ré-envoyée à l'objet `Printer`.

`KillDoc` ne peut annuler une impression déjà commencée, ni les tâches `PrintForm`.

Microsoft recommande de créer dans un module standard une sous-routine d'impression, qui pourra être appelée depuis toutes les applications disposant de ce module. Cette sous-routine poura aussi servir aux impressions de feuilles. Le code du Listing 13.3 contient deux arguments de type `Object`. Ce type de données permet de passer à la sous-routine aussi bien un objet `Form` qu'un objet `Printer`.

Listing 13.3 : Cette procédure permet d'imprimer les contrôles de la feuille

```
1:  Sub PrintAnywhere (Src As Object, Dest As Object)
2:    Dest.PaintPicture Src.Picture, Dest.Width / 2, Dest.Height / 2
3:    If TypeOf Dest Is Printer Then
4:      Printer.EndDoc
5:    End If
6:  End Sub
```

On peut ainsi imprimer une feuille contenant une image. Ou bien une feuille dans laquelle l'utilisateur entre des données qui seront par la suite envoyées à l'imprimante.

A la ligne 1, la sous-routine exige des arguments de source et de destination. La source esr toujours la feuille à imprimer. La destiantion peut être `Printer`. Lorsque vous êtes prêt à imprimer, vous appelez la procédure de cette façon :

```
Call PrintAnywhere (frmUserForm, Printer)  ' Print form.
```

La sous-routine imprime la feuille à l'aide de la méthode `PaintPicture`. `PaintPicture` dessine la feuille sur l'objet auquel elle est appliquée (la méthode). La méthode `PaintPicture` requiert trois valeurs : la forme à dessiner, la largeur de destination et la hauteur

de destination. Le code du Listing 13.3 dessine une feuille mesurant la moitié de la taille de la zone de destination.

L'instruction `If` s'assure que le destination est bien l'objet `Printer`, et non une autre feuille, et envoie la sortie à l'aide de la méthode `EndDoc`. (On aurait pu passer une deuxième feuille comme destination.)

L'instruction `If TypeOf` implique un nouveau genre de `If`. `If TypeOf... Is` permet d'interroger le type d'un objet.

En fait, l'instruction `If TypeOf... Is` fait plus que vérifier le type d'un objet, comme vous l'apprendrez au Chapitre 16.

Les propriétés *Font*

Le Tableau 13.3 présente certaines des propriétés de `Printer` qui permettent de spécifier la mise en forme du texte avant de l'envoyer à l'imprimante.

Tableau 13.3 : Les propriétés Font permettent de mettre le texte en forme avant d'imprimer

Propriété	*Description*
`Font`	Détermine la police à utiliser.
`FontBold`	Si la valeur est `True`, le texte sera imprimé en gras.
`FontCount`	Renvoie le nombre de polices supportées par l'imprimante.
`FontItalic`	Si la valeur est `True`, le texte sera imprimé en italique.
`FontName`	Contient le nom de la police utilisée.
`Fonts`	Contient la liste de toutes les polices installées sur le système. On accède à cette liste comme à un tableau de contrôle : `Fonts(0)` et `Fonts(FontCount - 1)` sont le premier et le dernier indice.
`FontSize`	Détermine la taille (en points) de la police utilisée.
`FontStrikeThru`	Si la valeur est `True`, le texte sera imprimé en barré.
`FontTransparent`	Si la valeur est `True`, le texte sera imprimé en transparent.
`FontUnderline`	Si la valeur est `True`, le texte sera imprimé en souligné.

Les attributs de police permettent d'enjoliver ou de mettre en valeur le texte imprimé. Le code suivant imprime les mots "Visual Basic" en gros caractères, gras et italiques :

```
Printer.FontItalic = True
Printer.FontBold = True
Printer.FontSize = 72      ' Corps 72.
Printer.Print "Visual Basic"
```

Les méthodes de `Printer`, nous l'avons dit, ne permettent pas seulement d'envoyer la sortie vers une imprimante, mais aussi vers une feuille. Examinez le Listing 13.4 et tentez de deviner ce qui apparaîtra sur la feuille. (Une sortie identique aurait pu être envoyée vers `Printer`, mais cet exemple permet de constater le résultat à l'écran.)

Listing 13.4 : Print permet également d'envoyer une sortie vers la feuille

```
1:  Private Sub cmdPrint_Click()
2:     ' Envoie une sortie vers la feuille
3:     ' à l'aide de la méthode Print.
4:     Dim intCtr As Integer
5:     Dim intCurX As Integer
6:     Dim intCurY As Integer
7:     '
8:     ' Définit les attributs de police.
9:       frmPrint.FontItalic = True
10:      frmPrint.FontBold = True
11:      frmPrint.FontSize = 36
12:    '
13:    ' Spécifie des mesures en twips.
14:      frmPrint.ScaleMode = vbTwips
15:    '
16:    ' Enregistre les positions X et Y (en twips)
17:    ' à chaque itération de la boucle.
18:    For intCtr = 1 To 10
19:      intCurX = frmPrint.CurrentX
20:      intCurY = frmPrint.CurrentY
21:      ' Texte noir et blanc en alternance.
22:      If (intCtr Mod 2) = 1 Then  ' Compteur de boucle.
23:         frmPrint.ForeColor = vbWhite
24:      Else
25:         frmPrint.ForeColor = vbBlack
26:      End If
27:      ' Affiche le texte.
28:      frmPrint.Print "Visual Basic"
29:      '
30:      ' Change les positions X et Y.
31:      frmPrint.CurrentX = intCurX + 350
32:      frmPrint.CurrentY = intCurY + 300
33:    Next intCtr
```

Listing 13.4 : Print permet également d'envoyer une sortie vers la feuille *(suite)*

```
34:  End Sub
35:
36:  Private Sub cmdExit_Click()
37:    End
38:  End Sub
```

Les lignes 9 à 11 spécifient le format du texte : corps 36, italique, gras. La ligne 14 spécifie que les positions `CurrentX` et `CurrentY` doivent être réglées en twips, et non en caractères. La ligne 18 commence la boucle qui affichera sur la feuille dix fois "Visual Basic".

Si vous modifiez ce code pour envoyer la sortie vers une imprimante, choisissez, à la ligne 23, une autre couleur que `vbWhite`. Si vous ne disposez pas d'une imprimante couleur, des valeurs comme `vbBlue` ou `vbRed` s'imprimeront de toute façon en niveaux de gris.

Au début de la boucle (lignes 19 et 20), le code stocke les valeurs courantes de `CurrentX` et `CurrentY`. Si ces valeurs n'étaient pas enregistrées, chaque méthode `Print` apparaîtrait sur une ligne pleine après la première méthode `Print`. En effet, chaque `Print` envoie automatiquement le curseur texte au début de la ligne suivante, à moins que l'on ne place un point-virgule à la fin de la méthode `Print`, ou que l'on ne règle les propriétés `CurrentX` et `CurrentY`, ce qui a été fait ici.

A la ligne 22, l'instruction `If` gère la couleur de la sortie. A chaque itération de la boucle, la couleur change du noir au blanc et vice versa. L'opérateur `Mod` renvoie `0` ou `1`, selon la valeur du compteur de boucle. Si `intCtr` est pair, `Mod` prend la valeur `0`, si `intCtr` est impair, `Mod` prend la valeur `1`.

C'est à la ligne 28 qu'a lieu l'affichage comme tel. La méthode `Print` devrait normalement envoyer le curseur texte à la ligne suivante, de sorte que les sorties `Print` subséquentes ne se surperposent pas. Mais les lignes 31 et 32 positionnent le curseur texte de sorte que chaque sortie morde un peu sur la précédente. Les lignes 31 et 32 ajoutent quelques twips aux valeurs `CurrentX` et `CurrentY` de la `Print` précédente (valeurs enregistrées aux lignes 19 et 20). A chaque itération de la boucle, les mots "Visual Basic" s'affichent dans une couleur différente et en surimpression partielle de la ligne précédente, et cela jusqu'à la dixième itération. La Figure 13.2 montre le résultat qui apparaît sur la feuille. Celui-ci aurait été le même si la sortie avait été envoyée vers l'objet `Printer`.

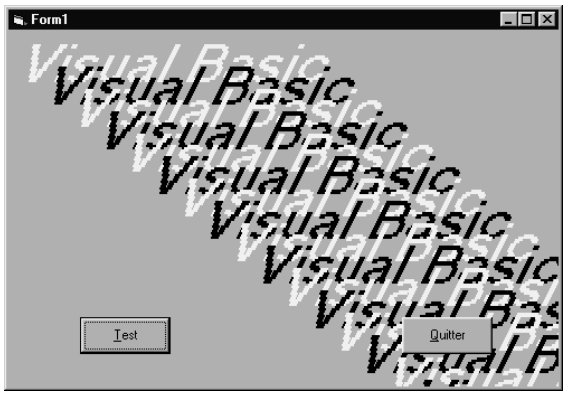


Figure 13.2
Les méthodes Print permettent d'envoyer du texte mis en forme.

Impression des feuilles

L'une des façons les plus simples d'envoyer des données à l'imprimante est d'imprimer une feuille. La méthode `PrintForm` peut s'appliquer à n'importe quelle feuille de votre projet. L'impression de feuilles relève des mêmes techniques que celles que nous avons étudiées pour la méthode `Print`.

Lorsque la méthode `PrintForm` est appliquée à une feuille, Visual Basic lance immédiatement l'impression. La feuille comme l'imprimante doivent donc être prêtes à ce moment. Cette section vous explique comment obtenir les meilleurs résultats avec `PrintForm`. La méthode `PrintForm` ne se prête pas à tous les types d'impression Visual Basic, mais suffit amplement pour l'impression de feuilles.

Le plus grand avantage de `PrintForm`, comme des autres fonctionnalités d'impression de Visual Basic, est que la sortie exploite les objets imprimante de Windows. Vous n'avez donc pas à vous soucier d'éventuelles spécificités liées à la marque ou au modèle du périphérique.

Voici le format de la méthode `PrintForm` :

```
[frmFormName.]PrintForm
```

`frmFormName` est optionnel. Si aucun nom de feuille n'est spécifié, Visual Basic applique automatiquement `PrintForm` à la feuille courante.

Pour imprimer, par exemple, une feuille nommée `frmAccPayable`, on insérerait dans la procédure événementielle ou le module approprié l'instruction suivante :

```
frmAccPayable.PrintForm     ' Imprime la feuille frmAccPayable.
```

Si `frmAccPayable` se trouve être la feuille courante (c'est-à-dire la feuille qui a le focus et dont la barre de titre est en surbrillance), le nom de la feuille source peut être omis :

```
PrintForm     ' Imprime la feuille frmAccPayable.
```

`Me` se réfère toujours à la feuille courante. Ainsi l'instruction suivante est-elle équivalente aux deux précédentes :

```
Me.PrintForm   ' Imprime la feuille frmAccPayable.
```

Comme nous l'avons vu au Chapitre 11, la méthode `Print` envoie du texte directement à la feuille. Vous pouvez donc envoyer la sortie vers la feuille avec `Print`, pour ensuite envoyer la feuille vers l'imprimer avec `PrintForm`. Gardez cependant à l'esprit que tous les contrôles de la feuille apparaîtront également sur la sortie imprimée.

Inconvénients de *PrintForm*

C'est la simplicté de `PrintForm` qui fait sa force. `PrintForm` est certainement la technique d'impression la plus utile et la plus simple que Visual Basic puisse offrir. Malheureusement, cette simplicité coûte quelques désavantages, dont vous devez être avisé.

Quelle que soit la résolution réelle de l'imprimante, `PrintForm` imprime toujours la feuille selon la résolution d'écran courante. Ce qui signifie, en général, un maximum de 96 dpi (*dots per inch*, points par pouce). La résolution des imprimantes atteignant en moyenne les 600 dpi, le rendu de la feuille imprimée sera nettement inférieur au rendu à l'écran. En effet, 96 dpi est une résolution tout à fait honnête pour un écran, mais tout à fait insuffisante pour une sortie papier de qualité.

Avant d'imprimer une feuille contenant des contrôles de type graphique ou tout autre élément visuel, vous devez vous assurer que la propriété `AutoRedraw` de la feuille est définie comme `True`. La valeur par défaut de `AutoRedraw` est `False`, ce qui implique que `Print` imprimera les contrôles graphiques directement au premier-plan. Lorsque `AutoRedraw` est `True`, l'image reste en arrière-plan pendant que `Print` fait son travail, de sorte qu'aucun élément ne se superpose à d'autres. Définie comme `False`, `AutoRedraw` peut permettre d'imprimer les images d'arrière-plan en premier. Vous pouvez ensuite imprimer

le texte par-dessus, mais il faudra redéfinir `AutoRedraw` comme `True` juste avant de lancer l'impression, afin que l'ensemble de la feuille sorte correctement.

Attention : pour que `PrintForm` puisse imprimer les objets placés sur la feuille au moment de la création (ainsi que les valeurs de contrôles initialisées lors de l'exécution, telles que le contenu des labels et autres zones de texte), `AutoRedraw` doit impérativement être `False`. Ainsi, si vous ajoutez des éléments graphiques en cours d'exécution, pour ensuite imprimer la feuille avec `PrintForm`, vous devrez définir comme `True` la propriété `AutoRedraw` de la feuille avant de procéder aux ajouts. Autrement, les éléments intégrés lors de l'exécution n'apparaîtraient pas sur la feuille imprimée.

L'impression peut être le fléau du programmeur Windows. Pour être certain des résultats, il faut tester l'application sur le plus grand nombre de modèles d'imprimantes possible. Il est intéressant, à cet effet, de distribuer votre programme à plusieurs utilisateurs-testeurs disposant d'imprimantes différentes. En fait, on ne peut jamais être certain du résultat pour toutes les sortes d'imprimantes, mais on obtient ainsi un échantillon significatif. La validité de tels tests dépend évidemment de ce que les pilotes de l'utilisateur soient correctement installés, et de ce que cet utilisateur sélectionne l'imprimante adéquate.

Il faut toutefois noter que le rôle de votre application, quant à la qualité des impressions, est toujours limité. En effet, c'est un peu là le "domaine réservé" de l'interface d'impression Windows. En se plaçant ainsi entre votre application et l'imprimante, Windows vous épargne d'ailleurs pas mal de codage. Les programmeurs MS-DOS, eux, devaient tenir compte de tous les types d'imprimantes existants — tâche sisyphéenne, puisque de nouveaux modèles devaient forcément sortir entre le moment où le programme était écrit et celui de sa distribution.

Dans le Listing 13.5, on envoie un message textuel vers une feuille, puis cette feuille vers l'imprimante.

Listing 13.5 : Affichage d'un message sur la feuille, puis impression de la feuille

```
1:  Dim blnAutoRedraw As Boolean   ' Contiendra la valeur de AutoRedraw.
2:  '
3:  frmBlank.Print "Répartition du matériel"
4:  frmBlank.Print    ' Blank line
5:  frmBlank.Print "Zone"; Tab(20); "Machines"
6:  frmBlank.Print "--------"; Tab(20); "--------"
7:  frmBlank.Print "Nord"; Tab(20); "Fraises"
8:  frmBlank.Print "Sud"; Tab(20); "Presses"
9:  frmBlank.Print "Est"; Tab(20); "Broyeurs"
10: frmBlank.Print "Ouest"; Tab(20); "Giboliniseurs"
```

Listing 13.5 : Affichage d'un message sur la feuille, puis impression de la feuille *(suite)*

```
11: '
12: ' Enregistre la valeur de AutoRedraw.
13: '
14: blnAutoRedraw = frmBlank.AutoRedraw
15: '
16: ' Imprime la feuille.
17: '
18: frmBlank.AutoRedraw = True
19: frmBlank.PrintForm
20: '
21: ' Restaure AutoRedraw.
22: '
23: frmBlank.AutoRedraw = blnAutoRedraw
```

Dans ce code, on enregsitre la valeur de `AutoRedraw` avant d'appliquer la méthode `PrintForm`. Dans un cas pareil, il est plus sûr de définir `AutoRedraw` comme `True` lors de la création (en supposant que la feuille ne recevra pas d'image ailleurs dans le programme). Mais vous pouvez recourir à cette sauvegarde de `AutoRedraw` avant d'imprimer une feuille.

Créez une procédure de module standard qui reçoit une feuille comme argument (les feuilles peuvent être passées au même titre que les variables), stocke la valeur de `AutoRedraw`, puis imprime la feuille à l'aide de `PrintForm`. Cette procédure généraliste vous épargnera de régler `AutoRedraw` pour chaque sortie `PrintForm`.

Avant toute impression, il convient de vérifier l'absence d'erreurs. L'imprimante pourrait être hors tension, non connectée ou ne plus avoir de papier. Le Listing 13.6 exploite à cet effet l'instruction `On Error Goto`.

Listing 13.6 : On Error Goto permet de gérer les erreurs d'impression

```
1:  Private Sub cmdPrintForm_Click ()
2:    Dim intBtnClicked As Integer
3:    On Error Goto ErrHandler    ' Définit le gestionnaire d'erreur.
4:    frmAccPayable.PrintForm     ' Imprime la feuille.
5:    Exit Sub
6:  ErrHandler:
7:    intBtnClicked = MsgBox("L'imprimante a un problème", vbExclamation,
      ➥"Erreur d'impression")
8:  End Sub
```

Vous pouvez imprimer une feuille sans la barre de titre, les icônes de contrôles et autres boutons Windows. Il suffit pour cela de définir comme `False` *les propriétés correspondantes.*

Avertir l'utilisateur

Il convient, avant de lancer une impression, d'en avertir l'utilisateur. Il a ainsi le temps d'allumer l'imprimante ou de charger le papier. Une simple boîte de dialogue lui permettra d'indiquer que l'imprimante est prête. D'inopportuns messages d'erreurs pourraient autrement apparaître, qui donneraient de votre application une image négative.

Le Listing 13.7 contient une procédure événementielle qui peut être adaptée aux besoins de l'application particulière. Cette procédure affiche la boîte de message reproduite en Figure 13.3. Cette boîte de message ne dit rien de très intéressant, mais laisse à l'utilisateur le temps de préparer l'imprimante avant le lancement de l'impression.

Listing 13.7 : Affiche une boîte de message avant de lancer l'impression

```
1:  Public Function PrReady() As Boolean
2:  ' Laisse à l'utilisateur le temps de se préparer.
3:    Dim intIsReady As Integer
4:    '
5:    ' L'utilisateur répond à la boîte de message
6:    ' pour indiquer qu'il est prêt.
7:    intIsReady = MsgBox("Veuillez préparer l'imprimante", vbOKCancel,
      ➥"Impression")
8:    '
9:    If (intIsReady = vbCancel) Then
10:      PrReady = False
11:   Else
12:      PrReady = True
13:   End If
14: End Function
```

Notez que la fonction déclarée doit renvoyer une valeur de type `Boolean`. L'appel de cette fonction peut donc être placé partout où un booléen valide est susceptible d'être reçu. Si l'utilisateur clique sur le bouton OK, la ligne 12 renvoie la valeur `True` à la fonction. Si, au contraire, l'utilisateur clique sur le bouton Annuler, la ligne 10 renvoie la valeur `False`. Le code interprète ces valeurs pour déterminer si l'impression doit avoir lieu ou non.

Le Listing 13.8 contient une procédure événementielle de bouton de commande qui peut servir à appeler la fonction `PrReady()`.

Figure 13.3
La boîte de message permet à l'utilisateur d'indiquer qu'il est prêt.

Listing 13.8 : Interroge la valeur de PrReady() avant d'imprimer

```
1:  Private Sub cmdPrint_Click()
2:     ' Imprime seulement si l'utilisateur
3:     ' indique qu'il est prêt.
4:     If PrReady() Then
5:        ' Appelle ReportPrint
6:     End If
7:  End Sub
```

La ligne 4 lance l'impression si et seulement si l'utilisateur répond à la fonction `PrReady()` en cliquant sur OK.

En résumé

Vous êtes maintenant capable d'envoyer une sortie vers l'imprimante. La gestion de l'impression est l'un des aspect les plus fastidieux de Visual Basic, car aucun contrôle spécifique n'est disponible pour assurer cette tâche. Pour obtenir des résultats satisfaisants, vous devez maîtriser les diverses méthodes de l'objet `Printer`. Cette technique a cependant l'avantage de vous laisser paramétrer exactement la sortie.

Les méthodes de `Printer` s'appliquent également aux objets `Form`. Vous pouvez ainsi envoyer du texte, enrichi de divers attributs de police, vers la feuille comme vers l'imprimante. Pour l'impression de feuilles, la méthode `PrintForm` est toutefois la plus indiquée.

Dans le chapitre suivant, vous découvrirez les fonctionnalités graphiques et multimédias de Visual Basic.

Questions-réponses

Q Que se passe-t-il quand deux applications envoient une sortie vers l'imprimante en même temps ?

R Heureusement, Windows se charge de gérer la priorité des sorties. Le PC ne peut faire réellement deux choses à la fois. Même si deux applications *semblent* envoyer simultanément deux sorties vers l'imprimante, l'une passe toujours avant l'autre. Windows place toutes les tâches d'impression dans la file d'attente et procède selon l'ordre d'envoi de ces tâches.

Q Ne vaut-il pas mieux envoyer d'abord toutes les sorties vers la feuille, puis cette feuille vers l'imprimante ?

R Les feuilles s'impriment à la résolution de l'écran, et non à celle de l'imprimante. La sortie sera de bien meilleure qualité si elle est envoyée directement à l'imprimante. En outre, les éléments devant apparaître sur la sortie imprimée ne doivent pas forcément apparaître sur la feuille. Ainsi, une application qui imprime des chèques sur du papier spécial n'aura aucun besoin de passer par une feuille pour chaque chèque. Un chèque, soit dit en passant, n'est pour le programme qu'une page comme une autre, et nécessite seulement que vous positionniez la sortie conformément aux différents "champs" du chèque papier.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Comment le programme peut-il déterminer le nombre d'imprimantes installées sur le système ?
2. L'instruction suivante déclare deux variables. Vrai ou faux ?

```
Dim intI As Integer, prnP As Printer
```

3. Quelle propriété permet de spécifier l'échelle des propriétés de l'objet `Printer` ?

4. Comment force-t-on le programme à imprimer sur une nouvelle page ?
5. Quelle instruction `If` spéciale permet de vérifier le type de données d'un objet ?
6. Aux procédures ne peuvent être passées que des variables. Vrai ou faux ?
7. `KillDoc` peut annuler n'importe quelle sortie imprimante, incluant celles qui ont été envoyées par `Printer.Print` et `PrintForm`. Vrai ou faux ?
8. Quelle référence abrégée renvoie à la feuille courante ?
9. A quelle résolution la méthode `PrintForm` imprime-t-elle généralement ?
10. Quelle valeur faut-il affecter à la propriété `AutoRedraw` avant d'imprimer une feuille ?

Exercices

1. Ecrivez une ligne de code qui imprime votre nom à la colonne 32.
2. **Chasse au bogue.** Yolande n'arrive pas à imprimer convenablement ses rapports. Elle a appris, il y a des années, le langage BASIC et vient de se mettre à la programmation Windows en Visual Basic. A l'époque, Yolande partait du principe qu'une page normale de texte — en environnement texte seul — contient exactement 66 lignes. Conséquemment, elle incrémente une variable compteur chaque fois qu'elle imprime une ligne. Lorsque le compteur atteint la valeur 66, on est censé être à la première ligne de la page suivante. Maintenant que Yolande est passée à Windows, sa logique ne fonctionne plus. Pourquoi ses rapports ne contiennent-ils plus exactement 66 lignes par page ?
3. Modifiez le Listing 13.2, qui recherche une imprimante couleur, en ajoutant une variable `Boolean` qui prendra la valeur `True` si une imprimante couleur est découverte. Si aucune imprimante couleur n'est trouvée, l'imprimante par défaut reste la même. La nouvelle variable booléenne indiquera au code subséquent si la boucle a ou non découvert une imprimante couleur. Faites du code une fonction qui renvoie une valeur `Boolean`.

Chapitre 14

Image et multimédia

Cercles, ovales, carrés... Ce n'est pas un retour à l'école maternelle que vous propose ce chapitre, mais une initiation aux fonctionnalités graphiques de Visual Basic. Vous découvrirez également ce qui distingue le contrôle Image du contrôle PictureBox. Enfin, vous apprendrez à créer des applications vivantes en intégrant son et vidéo.

Les ordinateurs d'aujourd'hui sont foncièrement multimédias. Le multimédia est impliqué dans tous les types de programmes, des progiciels aux applications familiales, en passant par les logiciels éducatifs. Plusieurs sociétés proposent des outils multimédias supplémentaires pour Visual Basic. Ces produits sont utiles au programmeur qui développe des applications hautement multimédias. Pour les autres, Visual Basic dispose d'un assortiment de contrôles multimédias internes tout à fait satisfaisant. Ce chapitre vous propose d'en découvrir l'essentiel.

Voici ce que nous étudierons aujourd'hui :

- les contrôles Image et PictureBox ;
- pourquoi le contrôle PictureBox est plus flexible que le contrôle Image ;
- les méthodes de dessin ;
- comment dessiner points, lignes, carrés, cercles et ellipses ;
- le contrôle multimédia et les périphériques qu'il supporte ;
- comment obtenir le statut d'un périphérique multimédia.

Les contrôles *Image* et *PictureBox*

Les contrôles Image et PictureBox ont essentiellement le même rôle. Ils permettent de placer des images sur la feuille, à partir de fichiers graphiques. Ils se distinguent seulement en ceci :

- Le contrôle PictureBox, plus flexible, supporte des méthodes et propriétés supplémentaires.
- Le contrôle Image est plus efficace pour les applications exécutées sur des machines lentes.

Vu la rapidité des ordinateurs modernes, vous n'observerez qu'exceptionnellement la supériorité de PictureBox sur Image. En fait, à moins que vous n'ayez à programmer pour des machines un peu fatiguées (comme en rencontre encore dans beaucoup d'entreprises, sans parler des administrations), le contrôle PictureBox est toujours le plus indiqué.

Les contrôles Image et PictureBox supportent tous deux les formats graphiques suivants :

- fichiers bitmaps, d'extension .BMP ;
- fichiers curseurs, d'extension .CUR ;
- fichiers GIF, d'extension .GIF extension ;
- fichiers icônes, d'extension .ICO ;
- fichiers JPEG, d'extension .JPEG ou .JPG ;
- métafichiers, d'extension .WMF ou .EMF ;
- fichiers d'extension .RLE.

Des fichiers relevant de plusieurs de ces formats apparaissent dans le dossier Graphics de Visual Basic que vous avez peut-être installé.

La propriété la plus importante, que se partagent les contrôles Image et PictureBox, est `Picture` qui contient l'image. Lors de la création, vous pouvez cliquer sur la propriété `Picture` de la fenêtre Propriétés pour choisir dans la boîte de dialogue Charger une image un fichier portant l'une des extensions supportées. Pour afficher une image lors de l'exécution, vous devez, à l'aide de la fonction `LoadPicture()`, associer le fichier graphique à la propriété `Picture` du contrôle concerné.

L'instruction suivante affecte une image à la propriété `Picture` d'un contrôle PictureBox :

```
picPortrait.Picture = LoadPicture("c:\Photos\Josette.wmf")
```

On ne peut affecter directement le chemin d'accès du fichier à la propriété `Picture`. C'est pourquoi la fonction `LoadPicture()` est la plus importante pour ce qui a trait aux images.

Voici le format complet de la fonction `LoadPicture()` :

```
LoadPicture([GraphicFileName] [,varSize] [,varColorDepth], [varX, varY])
```

Le premier argument, soit le nom du fichier, est optionnel. Si vous appelez la fonction `LoadPicture()` sans spécifier de nom de fichier, Visual Basic efface l'image du contrôle.

Le Tableau 14.1 présente les constantes nommées qui peuvent être employées pour l'argument *varSize*, si du moins vous le spécifiez. Cet argument définit la taille des icônes et curseurs. *varSize* est important, car les utilisateurs se servent souvent des paramètres d'affichage du Panneau de configuration pour fixer la taille des icônes et curseurs. L'argument *varSize* permet à votre programme d'exploiter ces valeurs système.

Tableau 14.1 : Constantes nommées pour l'argument varSize de la fonction LoadPicture()

Constante nommée	*Valeur*	*Description*
`vbLPSmall`	`0`	Petite icône système (varie selon la résolution d'écran).
`vbLPLarge`	`1`	Grande icône système (varie selon la résolution d'écran).
`vbLPSmallShell`	`2`	Déterminé par les paramètres d'affichage du Panneau de configuration. L'onglet Apparence présente les éléments qui seront affectées par cette valeur `varSize`.
`vbLPLargeShell`	`3`	Déterminé par les paramètres d'affichage du Panneau de configuration. L'onglet Apparence présente les éléments qui seront affectées par cette valeur `varSize`.
`vbLPCustom`	`4`	Les arguments `varX` et `varY` définissent la taille.

Le Tableau 14.2 présente les valeurs que peut prendre l'argument optionnel *varColorDepth* (profondeur de couleur pour les icônes et les curseurs).

Les arguments *varX* et *varY* sont requis si vous utilisez les valeurs de taille `vbLPSmallShell` ou `vbLPLargeShell`.

Sur la feuille, les contrôles Image et PictureBox agissent différemment, même si vous leur attribuez la même taille et leur affectez le même fichier graphique. Pour le contrôle Image, vous devez définir la propriété `Stretch` comme `True` avant de pouvoir régler les

Tableau 14.2 : Constantes nommées pour l'argument varColorDepth de la fonction LoadPicture()

Named Constant	*Value*	*Description*
`vbLPDefault`	`0`	Meilleure adéquation
`vbLPMonochrome`	`1`	2 couleurs
`vbLPVGAColor`	`2`	16 couleurs
`vbLPColor`	`3`	256 couleurs

propriétés `Width` et `Height`. Autrement, la taille du contrôle s'ajusterait à celle de l'image, et les propriétés `Width` et `Height` se modifieraient automatiquement. Lorsque vous placez un contrôle PictureBox sur la feuille, c'est l'image qui s'ajuste automatiquement aux mesures du contrôle. Ainsi, le contrôle PictureBox changera toujours la taille de son image pour répondre à vos spécifications, alors que le contrôle Image modifie ces mêmes valeurs tant que sa propriété `Stretch` n'est pas `True`.

La fonction `LoadPicture()` ne s'applique pas qu'aux contrôles Image et PictureBox, mais également aux feuilles. Au lieu d'une couleur, vous pouvez ainsi affecter à votre feuille une image d'arrière-plan. C'est ce que fait l'instruction suivante :

```
frmCheck.Picture = LoadPicture("Check.wmf")
```

Vous devez spécifier le chemin d'accès complet du fichier graphique affecté. Ce qui peut donner des instructions assez longues, par exemple :

```
frmCheck.Picture = LoadPicture("d:\Program Files\Microsoft Visual
Studio\Common\Graphics\metafile\business\Check.wmf")
```

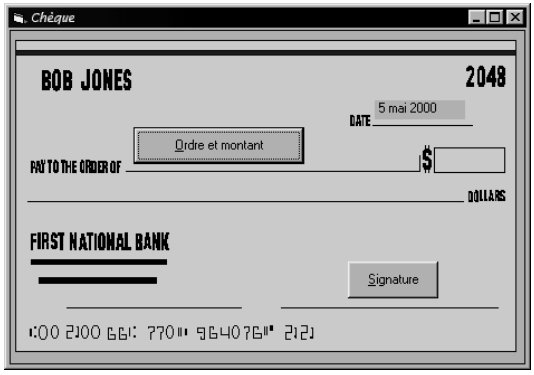
La Figure 14.1 montre la feuille résultante, avec labels et boutons de commande. L'image occupe tout l'arrière-plan. (Pour affecter une image d'arrière-plan lors de la création, utilisez la propriété `Picture`.)

Les contrôles de dessin

La Boîte à outils comprend deux outils de dessin :

- **Le contrôle Line.** Trace une droite entre les deux points que vous spécifiez.
- **Le contrôle Shape.** Trace la forme spécifiée dans les valeurs de propriétés.

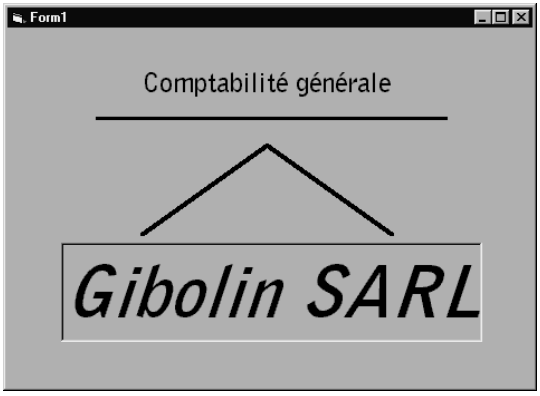
Figure 14.1
Une image peut être affectée à l'arrière-plan de la feuille.



Le contrôle *Line*

Le contrôle Line est utile à plusieurs égards, même dans une application qui n'exploite pas d'images. Les lignes permettent notamment de mettre en valeur les points importants d'une feuille. La Figure 14.2 illustre l'emploi de lignes pour une feuille qui s'affiche à chaque démarrage d'une application.

Figure 14.2
Les lignes aident à mettre en relief les informations de la feuille.



Lorsque vous double-cliquez sur le contrôle Line, Visual Basic affiche sur la feuille une ligne présentant à chaque extrémité des poignées de dimensionnement. Vous pouvez faire glisser ces poignées pour modifier la longueur de la ligne, et les déplacer à la verticale pour modifier l'angle. A mesure que vous modifiez la taille et les coordonnées de la ligne, Visual Basic actualise les valeurs de propriétés correspondantes.

Voici les propriétés importantes pour le contrôle Line :

- `BorderColor` définit la couleur de la ligne.
- `BorderStyle` définit le format de la ligne, à l'aide des valeurs présentées au Tableau 14.3 et illustrées par la Figure 14.3.

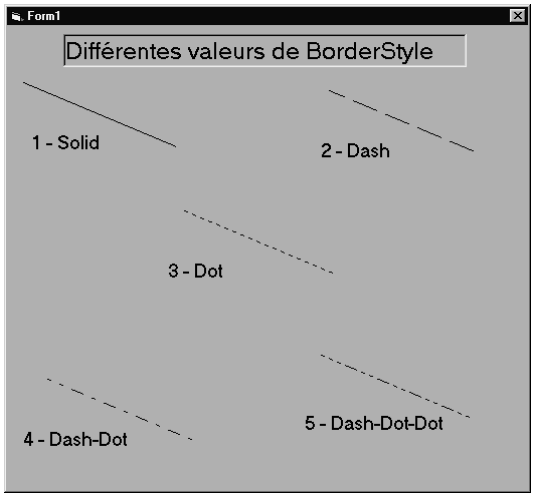


Figure 14.3
La propriété BorderStyle détermine l'aspect de la ligne.

- `BorderWidth` définit l'épaisseur (en points) de la ligne.
- `X1`, `Y1`, `X2` et `Y2` déterminent les coordonnées des deux extrémités de la ligne. A chaque point de la feuille correspondent deux valeurs, et la ligne est définie par deux points (les points entre lesquels elle apparaît).

Pour que `BorderStyle` puisse s'appliquer à la ligne, `BorderWidth` doit avoir la valeur `1`.

Le contrôle *Shape*

Alors que le contrôle Line ne trace que des lignes simples, le contrôle Shape permet de dessiner des formes diverses. C'est la propriété `Shape` qui définit cette forme. En voici les valeurs possibles :

- `0-Rectangle` dessine un rectangle.
- `1-Square` dessine un carré.

- `3-Circle` dessine un cercle.
- `4-Rounded Rectangle` dessine un rectangle aux coins arrondis.
- `5-Rounded Square` dessine un carré aux coins arrondis.

Tableau 14.3 : Valeurs de la propriété BorderStyle du contrôle Line

Constante nommée	*Description*
`0-Transparent`	L'arrière-plan de la feuille apparaît à travers la ligne.
`1-Solid`	Ligne pleine.
`2-Dash`	Ligne divisée en petits tirets.
`3-Dot`	Ligne pointillée.
`4-Dash-Dot`	Chaque tiret est suivi d'un point.
`5-Dash-Dot-Dot`	Chaque tiret est suivi de deux points.

Le Tableau 14.4 présente les autres propriétés susceptibles d'affecter les différentes formes tracées avec le contrôle Shape.

Tableau 14.4 : Propriétés affectant les formes dessinées avec le contrôle Shape

Propriété	*Description*
`BackStyle`	Si la valeur est `True`, la forme sera transparente et laissera voir l'arrière-plan de la feuille.
`BorderColor`	Définit la couleur de la bordure.
`BorderStyle`	Définit le style de la bordure (valeurs du Tableau 14.3).
`BorderWidth`	Définit l'épaisseur de la bordure (en twips).
`FillColor`	Définit la couleur du motif de remplissage (spécifié par la propriété `FillStyle`).
`FillStyle`	Définit le motif de remplissage. La Figure 14.4 illustre les huit motifs disponibles.
`Height`	Spécifie le point le plus haut de la forme.
`Width`	Spécifie l'axe le plus large de la forme.

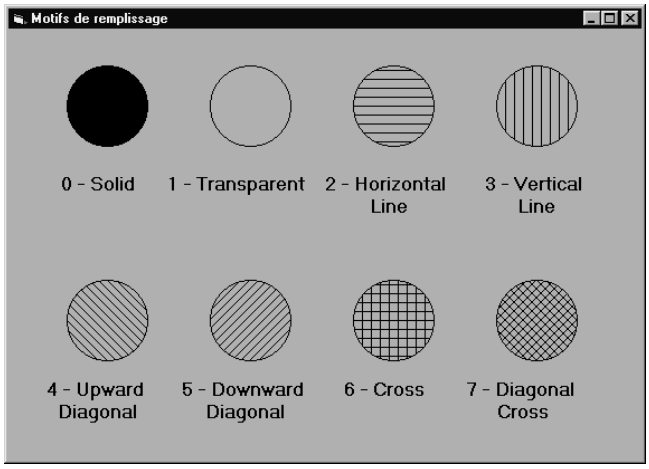


Figure 14.4
Les huit motifs disponibles pour FillStyle.

Pour illustrer le fonctionnement du contrôle Shape et de ses diverses propriétés, rien de mieux qu'une petite application. La Figure 14.5 montre le résultat de l'application que vous allez créer. Il s'agit de sélectionner, dans deux zones de liste, une forme et un motif de remplissage. La forme affichée en haut de la feuille se met à jour pour refléter les choix effectués.

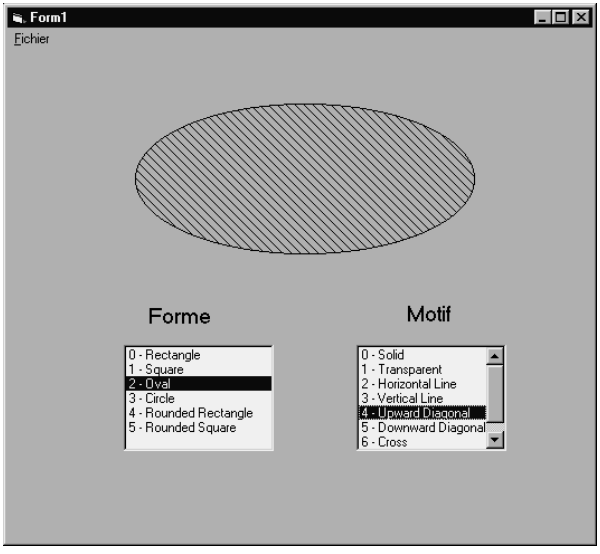


Figure 14.5
Sélectionnez une forme et un motif, et voyez le résultat.

Suivez ces étapes :

1. Créez une nouvelle application.
2. Placez sur la feuille les contrôles indiqués au Tableau 14.5.
3. Ajoutez le code du Listing 14.1 pour initialiser à l'exécution les deux zones de liste.
4. Lancez l'application et essayez diverses combinaisons de formes et de motifs à l'aide des zones de liste.

Tableau 14.5 : Contrôles et propriétés de la feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmShape`
Feuille	`Height`	`7005`
Feuille	`Left`	`105`
Feuille	`Top`	`105`
Feuille	`Width`	`7965`
Option de menu 1	`Name`	`mnuFile`
Option de menu 1	`Caption`	`&Fichier`
Option de menu 2	`Name`	`mnuFileExit` (un niveau seulement)
Option de menu 2	`Caption`	`&Quitter`
Shape	`Name`	`shpSample`
Shape	`Height`	`2025`
Shape	`Left`	`1710`
Shape	`Top`	`720`
Shape	`Width`	`4560`
Label 1	`Name`	`lblShape`
Label 1	`Caption`	`Forme`
Label 1	`Height`	`420`
Label 1	`Left`	`2160`

Tableau 14.5 : Contrôles et propriétés de la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Label 1	`Top`	`3390`
Label 1	`Width`	`1215`
Label 2	`Name`	`lblPattern`
Label 2	`Caption`	`Motif`
Label 2	`Height`	`420`
Label 2	`Left`	`5040`
Label 2	`Top`	`3360`
Label 2	`Width`	`1215`
List box 1	`Name`	`lstShape`
Zone de liste 1	`Height`	`1425`
Zone de liste 1	`Left`	`1560`
Zone de liste 1	`Top`	`3960`
Zone de liste 1	`Width`	`2010`
Zone de liste 2	`Name`	`lstPattern`
Zone de liste 2	`Height`	`1425`
Zone de liste 2	`Left`	`4680`
Zone de liste 2	`Top`	`3960`
Zone de liste 2	`Width`	`2010`

Listing 14.1 : Initialise les zones de liste et répond aux sélections de l'utilisateur

```
1:  Private Sub Form_Load()
2:    ' Initialise la liste déroulante Forme.
3:    lstShape.AddItem "0 - Rectangle"
4:    lstShape.AddItem "1 - Square"
5:    lstShape.AddItem "2 - Oval"
6:    lstShape.AddItem "3 - Circle"
7:    lstShape.AddItem "4 - Rounded Rectangle"
```

```
8:    lstShape.AddItem "5 - Rounded Square"
9:
10:   ' Initialise la liste déroulante Motif.
11:   lstPattern.AddItem "0 - Solid"
12:   lstPattern.AddItem "1 - Transparent"
13:   lstPattern.AddItem "2 - Horizontal Line"
14:   lstPattern.AddItem "3 - Vertical Line"
15:   lstPattern.AddItem "4 - Upward Diagonal"
16:   lstPattern.AddItem "5 - Downward Diagonal"
17:   lstPattern.AddItem "6 - Cross"
18:   lstPattern.AddItem "7 - Diagonal Cross"
19:
20:   ' Définit la première valeur de chaque liste comme valeur par défaut.
21:   lstShape.ListIndex = 0
22:   lstPattern.ListIndex = 0
23:
24: End Sub
25:
26: Private Sub lstPattern_Click()
27:   ' Change le motif en fonction de la sélection.
28:   shpSample.FillStyle = lstPattern.ListIndex
29: End Sub
30:
31: Private Sub lstShape_Click()
32:   ' Change la forme en fonction de la sélection.
33:   shpSample.Shape = lstShape.ListIndex
34: End Sub
35:
36: Private Sub mnuFileExit_Click()
37:   End
38: End Sub
```

Ni le contrôle Line ni le contrôle Shape ne supportent les procédures événementielles.

Les méthodes de dessin

Les fonctionnalités de dessin de Visual Basic ne se limitent naturellement pas aux contrôles Line et Shape. Il est possible de manipuler individuellement les pixels de la feuille pour dessiner une image point par point. La méthode `PSet` qui s'applique aux feuilles, permet de dessiner des lignes, des cadres et des cercles sans l'aide des contrôles. Les lignes et formes dessinées en cours d'exécution par les contrôles Line et Shape ne donnent pas de très bons résultats. Les méthodes de dessin offrent beaucoup plus de précision.

Les méthodes de dessin peuvent être appliquées aux contrôles de la feuille aussi bien qu'à la feuille elle-même.

La méthode `PSet` affiche ou masque individuellement les pixels de la feuille. Voici le format de `PSet` :

```
frmName.PSet [Step] (intX, intY) [color]
```

A moins que l'on ne spécifie une autre échelle pour les propriétés `ScaleX` et `ScaleY`, l'intersection ligne/colonne `0, 0` (soit *intX* et *intY*) correspond au premier pixel du coin supérieur gauche de la feuille. L'instruction suivante provoque l'affichage du pixel situé à 100 pixels à droite et 200 pixels en bas de l'origine (coin supérieur gauche) :

```
frmDraw.PSet (100, 200)    ' Affiche un pixel.
```

Le pixel affiché prend par défaut la couleur spécifiée de la propriété `ForeColor` de la feuille (ou du contrôle PictureBox). On peut définir une nouvelle couleur à l'aide d'une valeur hexadécimale, ou de l'une de ces constantes nommées `vbBlack`, `vbRed`, `vbGreen`, `vbYellow`, `vbBlue`, `vbMagenta`, `vbCyan` et `vbWhite`.

L'instruction suivante masque le pixel situé à la position `(100, 200)` en lui affectant la couleur employée pour l'arrière-plan de la feuille :

```
frmDraw.PSet (100, 200) frmDraw.BackColor   ' Masque un pixel.
```

Si à cette méthode `PSet` succédait la suivante, un autre pixel apparaîtrait à la position `(300, 350)` :

```
frmDraw.PSet (300, 350)    ' Affiche un pixel
```

L'option `Step` de la méthode `PSet` change la position des pixels subséquents. Après le premier appel de la méthode `PSet`, l'option `Step` rend *relatives* les valeurs `intX` et `intY` de la prochaine méthode `PSet`. Ainsi, si l'on avait ajouté à la méthode `PSet` précédente l'option `Step`, le pixel serait apparu 300 pixels à droite et 350 pixels en bas du premier pixel dessiné. C'est justement ce que fait l'instruction suivante :

```
frmDraw.PSet Step (300, 350)    ' Position relative.
```

On peut également tracer des lignes en incluant `PSet` dans une boucle :

```
For intX = 1 to 100
  frmDraw.PSet (intX, 250)
Next intX
```

Ne spécifiez jamais de coordonnées qui dépassent les limites réelles de la feuille. Cela générerait une erreur d'exécution.

Plutôt que de tracer une ligne pixel par pixel, ou à l'aide d'une boucle, utilisez la méthode `Line`. Elle est faite pour ça. En voici le format :

```
frmName.Line [Step] (intX1, intY1) - [Step] (intX2, intY2), [Color] [B][F]
```

Les deux paires de coordonnées définissent les pixels de début et de fin de la ligne. L'option `Step` transforme les coordonnées qu'elle précède en coordonnées relatives, c'est-à-dire relatives au dernier point dessiné.

La méthode suivante trace une ligne du pixel `100, 100` au pixel `150, 150` :

```
frmDraw.Line (100, 100) - (150, 150)
```

Comme dans la méthode `PSet`, la valeur optionnelle *`Color`* permet de spécifier une couleur, à l'aide d'une valeur hexadécimale ou d'une constante nommée. Si vous ne spécifiez pas de couleur, Visual Basic emploie pour la ligne la couleur d'arrière-plan de la feuille.

L'option `B` permet de dessiner un cadre avec la méthode `Line` :

```
frmDraw.Line (100, 100) - (150, 150), , B
```

Les deux virgules doivent impérativement être incluses, même si *`Color`* n'est pas spécifié. Cela afin que Visual Basic reconnaisse `B` comme l'option "cadre". Les deux paires de coordonnées spécifient la position des coins supérieur gauche et inférieur droit du cadre.

Les paires de coordonnées définissent les extrémités de la lignes et les coins opposés du cadre. Les lignes peuvent être dirigées vers le haut comme vers le bas ; la première paire de coordonnées peut désigner un point situé en bas ou à droite de la seconde paire de coordonnées.

Si l'option `F` est spécifiée, le cadre sera rempli de la même couleur que celle qui a été définie pour le contour :

```
frmForm.Line (35, 40) - (150, 175), vbGreen, BF  ' Cadre rempli de vert.
```

Si vous dessinez un cadre plus grand que la feuille, Visual Basic le tronque et affiche ce qui est affichable. Même si vous agrandissez la feuille, le cadre ne sera pas affiché en entier. Il faut le redessiner.

Le Listing 14.2 contient une procédure événementielle de bouton de commande. Cette procédure dessine sur la feuille une série de cadres, depuis le coin supérieur droit jusqu'au coin inférieur gauche. La Figure 14.6 montre le résultat.

Listing 14.2 : Dessin d'un motif à l'aide de la méthode Line, option "cadre"

```
1:  Private Sub cmdBoxes_Click()
2:    Dim intStartX As Integer
3:    Dim intStartY As Integer
4:    Dim intLastX As Integer
5:    Dim intLastY As Integer
6:    Dim intCtr As Integer
7:
8:    intStartX = 0
9:    intStartY = 0
10:   intLastX = 1000
11:   intLastY = 800
12:
13:   For intCtr = 1 To 20
14:     frmBoxes.Line (intStartX, intStartY)-(intLastX, intLastY), , B
15:
16:     ' prépare la position des prochains cadres.
17:     intStartX = intStartX + 400
18:     intStartY = intStartY + 400
19:     intLastX = intLastX + 400
20:     intLastY = intLastY + 400
21:   Next intCtr
22:
23: End Sub
```

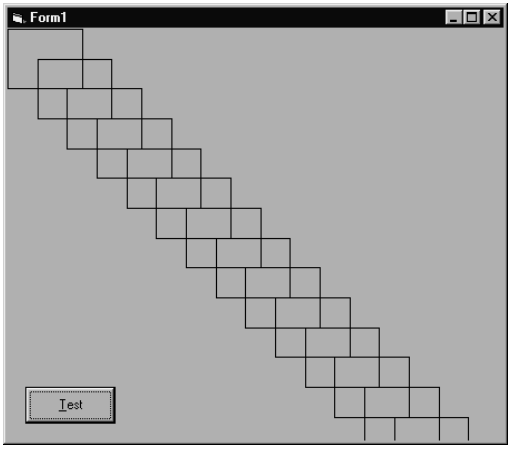


Figure 14.6
La méthode Line permet aussi de dessiner des cadres.

Il existe également des méthodes pour dessiner des formes courbes. Voici le format de la méthode `Circle`, qui permet de dessiner des cercles et des ellipses :

```
frmDraw.Circle [Step] (intX, intY), sngRadius, [Color], , , ,
➥sngAspect
```

Une ellipse est un cercle allongé (ce que l'on appelle couramment, et à tort, un ovale).

Dans le format indiqué ci-haut, les quatre virgules correspondent à des arguments de `Circle` que nous ne présenterons pas ici. Ces virgules doivent cependant être incluses si vous spécifiez l'argument `sngAspect`.

Un cercle n'ayant pas d'extrémités, les coordonnées `intX` et `intY` désignent le centre. `sngRadius` spécifie le rayon (en pixels, sauf valeur contraire de `ScaleMode`). Le mot clé optionnel `Step` détermine si les coordonnées du centre sont relatives à un autre objet dessiné.

L'instruction suivante dessine un cercle dont le centre se situe à 300 pixels du bord gauche de la feuille et à 200 pixels du bord supérieur, et dont le rayon mesure 100 pixels :

```
frmDraw.Circle (300, 200), 100
```

Tant qu'une autre valeur `Color` n'est pas spécifiée, la méthode `Circle` emploie la couleur d'arrière-plan de la feuille.

Pour dessiner une ellipse, il faut spécifier la valeur de l'argument `sngAspect`, qui détermine le ratio d'aspect. En fonction de la valeur, le cercle sera étiré selon l'axe horizontal X (valeur `sngAspect` inférieure à `1`), ou dans l'axe vertical Y (valeur `sngAspect` supérieure à `1`). Une valeur `sngAspect` de `1` donne un cercle parfait.

Le ratio d'aspect définit la forme de l'ellipse. Il s'agit en fait d'une mesure hauteur/largeur qui partirait du centre. Le ratio d'aspect multiplie le rayon dans chaque direction. Un ratio d'aspect de `4`, par exemple, donne une ellipse quatre fois plus haute que large ; un ratio d'aspect de `4/10/2` (ou `.2`) donne une ellipse cinq fois plus large que haute.

L'instruction suivante dessine sur la feuille l'ellipse reproduite en Figure 14.7 :

```
frmDraw.Circle (1000, 1250), 400, , , , (4 / 10 / 2)
frmDraw.Circle (1750, 1250), 400, , , , 4
```

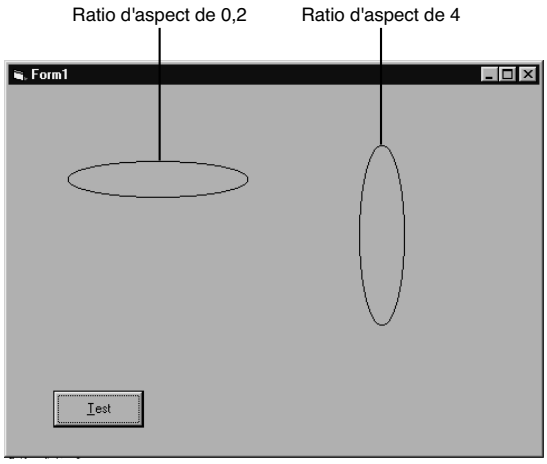


Figure 14.7
Le ratio d'aspect détermine la forme de l'ellipse.

Le contrôle multimédia

Malgré sa puissance, le contrôle multimédia est très simple d'utilisation. Il nécessite un minimum de code. Le contrôle multimédia permet d'incorporer des objets correspondant aux périphérique multimédias *simples* suivants :

- lecteur de CD audio (`CDAudio`) ;
- lecteur de DAT (`DAT`) ;
- vidéo numérique (`DigitalVideo`) ;
- overlay (`Overlay`) ;
- scanner (`Scanner`) ;
- magnétoscope (`Vcr`) ;
- lecteur de vidéodisques (`Videodisc`) ;
- autres périphériques avec pilote fourni (`Other`).

Entre parenthèses figurent les valeurs que peut prendre la propriété `DeviceType` du contrôle multimédia (voir plus loin).

Visual Basic supporte aussi les objets représentant les périphériques multimédias *composites* suivants :

- lecteur/enregistreur audio (fichiers .WAV) ;

- séquenceur MIDI (fichiers .MID) ;
- lecteur/enregistreur vidéo (fichiers .AVI).

Les périphériques multimédias simples n'exigent pas qu'un fichier soit associé au contrôle. Les CD audio, par exemple, ne requièrent pas de fichier spécifique. Les périphériques multimédias composites nécessitent un fichier externe. Par exemple, un lecteur audio doit lire un fichier .WAV précis.

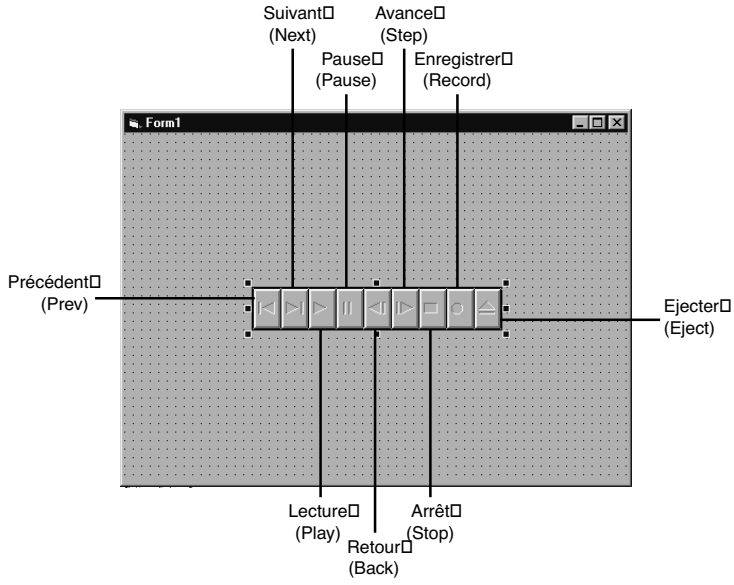
Evidemment, le PC sur lequel s'exécute votre application multimédia doit reconnaître les périphériques multimédias standards pour le son, les images, la vidéo, etc., ainsi que disposer d'un lecteur de CD-ROM (ou périphérique compatible, tel qu'un DVD).

Exploitation du contrôle multimédia

Comme le contrôle Common Dialog, le contrôle multimédia n'est pas inclus dans la Boîte à outils standard. Il faut donc l'ajouter. Appuyez sur Ctrl-T pour ouvrir la boîte de dialogue Composants, puis sélectionnez l'option Microsoft Multimedia Control 6.0.

Lorsque vous placez le contrôle multimédia sur la feuille, un jeu de boutons apparaît, semblables à ceux que l'on trouve sur les magnétoscopes ou radiocassettes (voir Figure 14.8).

Figure 14.8
Le contrôle multimédia sur la feuille.



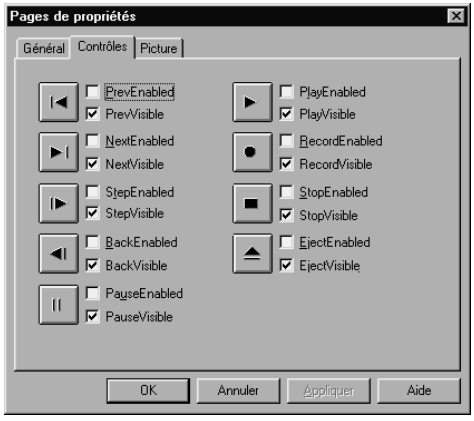
Le contrôle multimédia est un contrôle intelligent qui reconnaît les possibilités du périphérique qui lui est associé. Ainsi, le bouton Lecture ne sera pas disponible si vous avez ejecté le CD. Naturellement, vous pouvez choisir les boutons qui apparaissent, par le biais de diverses propriétés. Par exemple, les CD audio ne pouvant être enregistrés, il convient de masquer le bouton Enregistrer, plutôt que de simplement le désactiver.

Le contrôle multimédia renseigne votre application sur le périphérique et ses paramètres courants. L'application peut ainsi afficher le nombre de pistes contenues sur un CD audio.

Le contrôle multimédia contient une liste de périphériques aux boutons présélectionnés, de sorte que vous n'avez pas à définir la configuration de chaque média spécifique. Le contrôle multimédia exploite la propriété `DeviceType` qui, lorsqu'on lui affecte la valeur correspondant au périphérique désiré (par exemple `CDAudio` pour faire du contrôle multimédia un lecteur CD), affiche automatiquement les boutons requis. Toutes les valeurs possibles de `DeviceType` ont été données dans la liste des périphériques supportés, au début de la section précédente. Il suffit d'indiquer au contrôle multimédia ce que vous voulez qu'il soit, et il se configure lui-même.

Le contrôle multimédia dispose lui aussi de ses Pages de propriétés (que l'on obtient en double-cliquant sur la propriété `(Personnalisé)` dans la fenêtre Propriétés), qui facilitent le réglage. Ainsi, à l'onglet Contrôles, vous pouvez sélectionner les boutons qui doivent apparaître, ainsi que les activer ou les désactiver (voir Figure 14.9).

Figure 14.9
Les Pages de propriétés du contrôle multimédia facilitent le réglage des diverses options, par exemple les boutons.



Lorsque vous placez le contrôle multimédia sur la feuille, les boutons sont tous affichés, mais tous désactivés. Ils le restent tant que vous ne sélectionnez pas une valeur `DeviceType`.

Lecteur de CD audio

Pour exploiter un CD audio depuis une application Visual Basic, il suffit de placer sur une feuille le contrôle multimédia, et d'en régler la propriété `DeviceType` comme `CDAudio`. Le jeu de boutons adéquat apparaît automatiquement : vous avez votre lecteur CD. Libre à vous d'ajouter quelques détails, par exemple un label qui affiche les pistes.

Le contrôle multimédia se met à jour chaque fois qu'un événement `StatusUpdate` a lieu. Par exemple, pour un `DeviceType CDAudio`, `StatusUpdate` suit chaque changement de piste, ainsi que le début et l'arrêt de la lecture. Le label des pistes pourra ainsi être adéquatement initialisé par une procédure événementielle de `StatusUpdate`.

Le langage de commandes du contrôle multimédia

Le contrôle multimédia supporte son propre langage de commandes miniature (rien à voir, bien sûr, avec un vrai langage de programmation). La propriété `Command` peut prendre pour valeurs des commandes en un seul mot. Le Tableau 14.6 décrit ces commandes.

Tableau 14.6 : Valeurs de la propriété Command du contrôle multimédia

Command	*Description*
`Back`	Recule d'une piste.
`Close`	Ferme le périphérique.
`Eject`	Ejecte le Cd du lecteur de CD-ROM.
`Next`	Passe au début de la piste suivante (au début de la dernière piste si c'est la piste en cours de lecture).
`Open`	Ouvre le périphérique.
`Pause`	Suspend la lecture.
`Play`	Lance la lecture.
`Prev`	Revient au début de la piste lue. Si deux commandes `Prev` ont lieu en moins de trois secondes, revient au début de la piste précédente (au début de la première piste si c'est la piste en cours de lecture).
`Record`	Initialise l'enregistrement.
`Save`	Enregistre le fichier ouvert.

Tableau 14.6 : Valeurs de la propriété Command du contrôle multimédia *(suite)*

Command	*Description*
`Seek`	Recherche une piste vers l'avant ou vers l'arrière. (On préfère en général `Next` et `Prev`, en raison de l'ambiguïté directionnelle de `Seek`.)
`Stop`	Arrête la lecture.
`Step`	Avance d'une piste.

Le code peut modifier la valeur de la propriété `Command` en cours d'exécution, et le contrôle multimédia répondra adéquatement.

Le contrôle multimédia peut se passer totalement des interventions de l'utilisateur. Il suffit de masquer tous les boutons et de piloter le périphérique depuis le code, à l'aide de la propriété `Command`.

Création d'un lecteur de CD

Maintenant que nous vous avons présenté le contrôle multimédia, vous allez l'utiliser pour concevoir un lecteur de CD audio. Créez un nouveau projet et adaptez feuille, contrôles et propriétés aux spécifications du Tableau 14.7.

Tableau 14.7 : Propriétés et valeurs à utiliser pour la feuille du contrôle multimédia

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmCD`
Feuille	`Caption`	`Lecteur CD`
Feuille	`Height`	`3600`
Feuille	`Width`	`4800`
Label 1	`Name`	`lblCD`
Label 1	`Alignment`	`2-Center`
Label 1	`BorderStyle`	`1-Fixed Single`
Label 1	`Caption`	`Lecteur CD`

Tableau 14.7 : Propriétés et valeurs à utiliser pour la feuille du contrôle multimédia *(suite)*

Contrôle	*Propriété*	*Valeur*
Label 1	`Font style`	`BoldUnderline`
Label 1	`Font size`	`18`
Label 1	`Height`	`495`
Label 1	`Left`	`1320`
Label 1	`Top`	`480`
Label 1	`Width`	`1935`
Label 2	`Name`	`lblTrack`
Label 2	`Alignment`	`1-Right Justify`
Label 2	`Caption`	`Piste :`
Label 2	`Font style`	`Bold`
Label 2	`Font size`	`12`
Label 2	`Height`	`255`
Label 2	`Left`	`1200`
Label 2	`Top`	`2280`
Label 2	`Width`	`1215`
Label 3	`Name`	`lblTrackNum`
Label 3	`Caption`	(vide)
Label 3	`Font style`	`Bold`
Label 3	`Font size`	`12`
Label 3	`Height`	`375`
Label 3	`Left`	`2520`
Label 3	`Top`	`2280`

Tableau 14.7 : Propriétés et valeurs à utiliser pour la feuille du contrôle multimédia *(suite)*

Contrôle	*Propriété*	*Valeur*
Label 3	`Width`	`615`
Contrôle multimédia	`Name`	`mmcCD`
Contrôle multimédia	`DeviceType`	`CDAudio`

Une fois les contrôles placés et configurés, saisissez le code du Listing 14.3.

Listing 14.3 : Gestion du lecteur CD

```
1:  Private Sub Form_Load()
2:    ' Ouvre le CD.
3:    mmcCD.Command = "Open"
4:  End Sub
5:
6:  Private Sub Form_Unload(Cancel As Integer)
7:    ' Réinitialise le contrôle multimédia.
8:    mmcCD.Command = "Close"
9:  End Sub
10:
11: Private Sub mmcCD_StatusUpdate()
12:   ' Met à jour le label de pistes.
13:   lblTrackNum.Caption = mmcCD.Track
14: End Sub
```

Au premier chargement de la feuille, l'application ouvre le lecteur CD. La ligne 8 vide le périphérique de la mémoire avant que l'application ne se ferme (au même titre que les fichiers, il faut toujours refermer un périphérique multimédia). La ligne 13 met à jour le numéro de piste à chaque changement d'état (`StatusUpdate`) du CD. La valeur de la propriété `UpdateInterval` du contrôle multimédia spécifie l'intervalle entre chaque mise à jour (la valeur par défaut est `1000`, c'est-à-dire mille millisecondes).

La Figure 14.10 montre le résultat. L'application est simple, mais elle marche ! Vous pourriez l'améliorer encore en ajoutant une option de menu Fichier, Quitter, ainsi qu'un gestionnaire d'erreurs (voir section suivante).

Certains des boutons désactivés, tels le bouton Enregistrer, sont absolument inutiles dans cette application. Vous pouvez les cacher pour faire un peu de ménage.

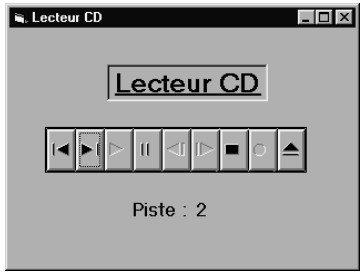


Figure 14.10
Ce lecteur CD tout à fait fonctionnel ne vous a pas coûté beaucoup de peine !

Vous ne maîtrisez pas seulement la fonction lecteur CD du contrôle multimédia, mais aussi tous les autres périphériques supportés ! Comme vous le verrez dans les sections suivantes, les autres types de périphériques fonctionnent de la même manière.

Notification et gestion d'erreurs

Lors de l'exécution du contrôle multimédia, plusieurs vérifications d'erreurs sont effectuées, afin que la lecture se fasse sans accrocs.

La propriété `Notify` déclenche un événement `Done` chaque fois que la commande *suivante* du contrôle multimédia s'exécute correctement. Pour lancer automatiquement une commande lorsque la commande précédente est terminée, vous pouvez définir la propriété `Notify` comme `True`. Définie comme `True`, la propriété `Wait` garantit que le contrôle ne renviendra pas à l'application avant que la commande précédente du contrôle multimédia ne se soit correctement exécutée. Le code suivant permet de vérifier si une erreur a eu lieu après envoi d'une commande :

```
If (frmCD.mmcCD.Error) then
  intMsg = MsgBox("Problème avec le CD", vbCritical)
Else
   ' Le code continue et lit le CD.
```

Mieux vaut ne pas placer d'instruction `On Error Goto` au début de la procédure, car vous ne pourriez alors savoir quelle commande précise a généré l'erreur. En faisant suivre chaque commande d'une routine de vérification d'erreur, vous assurez un meilleur suivi des problèmes et de leur signification.

Le Tableau 14.8 présente plusieurs valeurs de la propriété `Mode`. La propriété `Mode` permet d'interroger divers états, ou *modes*, du contrôle multimédia lors de l'exécution de l'application.

Tableau 14.8 : Valeurs de la propriété Mode du contrôle multimédia

Constantes nommées	*Description*
`mciModeNotOpen`	Périphérique non ouvert
`mciModeStop`	Périphérique arrêté
`mciModePlay`	Périphérique en cours de lecture
`mciModeRecord`	Périphérique en cours d'enregistrement
`mcuModeSeek`	Périphérique en cours de recherche
`mciModePause`	Périphérique mis en pause
`mciModeReady`	Périphérique prêt

Création d'un lecteur WAV

Les fichiers WAV sont des fichiers stockés sur l'ordinateur. Pour les lire, un périphérique multimédia composite est requis. Les périphériques multimédias composites obtiennent les données d'un fichier, et non d'une source extérieure telle qu'un CD audio. Votre PC contient beaucoup de fichiers WAV. C'est de ce type de fichiers que le Panneau de configuration se sert pour associer des sons aux événements système.

Cette section vous explique comment créer un lecteur de fichier WAV. Vous pourrez ainsi mettre en œuvre les valeurs de `Mode` présentées au Tableau 14.8. Nous nous servirons pour cette application du fichier Windows TADA.WAV, situé dans le dossier Windows\Media.

S'il est toujours ouvert, enregistrez puis fermez le projet lecteur CD. Il servira de base à ce nouveau projet. Enregistrez-le donc sous un nouveau nom, puis modifiez-le en suivant ces étapes :

1. Renommez le label du haut `lblWav` et définissez la propriété `Caption` comme `Lecteur WAV`. Définissez `Width` comme `2415`.
2. Renommez la feuille `frmWav` et définissez sa `Caption` comme `Lecteur audio WAV`.
3. Donnez au contrôle multimédia le nom `mmcWav` et changez sa propriété `DeviceType` en `WaveAudio`. Vous pouvez changer les autres propriétés depuis la fenêtre Propriétés ou dans les Pages de propriétés. Affectez à `Filename` le chemin d'accès du fichier TADA.WAV.
4. Les fichiers WAV ne contiennent pas de pistes, mais le label va quand même nous servir. Renommez le label de gauche (`lblTrack`) en `lblStatus` et définissez sa propriété `Caption` comme `Etat`.

5. Renommez le label de droite (`lblTrackNum`) en `lblStatusValue` et videz la `Caption`. Définissez une `Width` de `2565`.
6. Ajoutez deux labels en-dessous des précédents, et attribuez-leurs les mêmes propriétés `Width` et `Font`. (Vous pouvez les copier pour créer un jeu de tableaux de labels.) Nommez le premier lable `lblFile` (à moins que vous n'ayez créé un tableau de contrôle) et définissez sa `Caption` comme `Fichier :`. Il faudra sans doute l'agrandir.
7. Nommez le label de droite `lblFileValue` (à moins que vous n'ayez créé un tableau de contrôle). Laissez la propriété `Caption` vide. Centrez les labels sous les boutons. Le résultat devrait ressembler à la Figure 14.11.

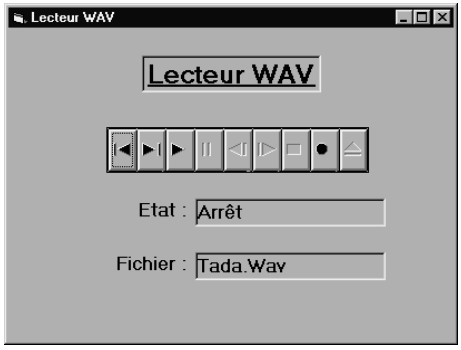


Figure 14.11
Notre lecteur WAV est presque terminé.

Modifiez la procédure événementielle `Form_Load()` comme ceci :

```
Private Sub Form_Load ()
  ' Ouvre le lecteur WAV.
  mmcWAV.Command = "Open"
End Sub
```

Vous devez également modifier la procédure événementielle `Form_Unload()` :

```
Private Sub Form_Unload(Cancel As Integer)
  ' Referme le lecteur WAV.
  mmcWAV.Command = "Close"
  Unload Me  ' Décharge aussi la feuille.
End Sub
```

Exécutez l'application. Lorsque vous cliquez sur le bouton Lecture, un son se fait entendre. Cliquez sur Retour pour l'entendre de nouveau.

Le lecteur WAV n'est pas encore tout à fait terminé. Les labels situés sous les boutons n'affichent pas encore l'état ni le nom du fichier. Initialisez ces labels dans la procédure événementielle `StatusUpdate()` (voir Listing 14.4). Vous devrez utiliser les valeurs de `Mode`.

Listing 14.4 : Initialise les labels avec les informations d'état

```
1:  Private Sub mciWAV_StatusUpdate()
2:    ' Affiche l'état.
3:    If mmcWAV.Mode = mciModeNotOpen Then
4:      lblStatusValue(0).Caption = "Non prêt"
5:    ElseIf mmcWAV.Mode = mciModeStop Then
6:      lblStatusValue(0).Caption = "Arrêt"
7:    ElseIf mmcWAV.Mode = mciModePlay Then
8:        lblStatusValue(0).Caption = "Lecture"
9:    ElseIf mmcWAV.Mode = mciModeRecord Then
10:     lblStatusValue(0).Caption = "Enregistrement"
11:   ElseIf mmcWAV.Mode = mciModePause Then
12:     lblStatusValue(0).Caption = "Pause"
13:   ElseIf mmcWAV.Mode = mciModeReady Then
14:     lblStatusValue(0).Caption = "Prêt"
15:   End If
16:   ' Affiche le nom du fichier lu.
17:   lblStatusValue(1).Caption = mmcWAV.FileName
18: End Sub
```

La procédure événementielle `StatusUpdate` *présuppose que vous ayez créé un tableau de contrôle pour les labels. Si vous ne l'avez pas fait, modifiez le nom des labels selon les noms que vous avez employés. Sinon, l'application ne pourra s'exécuter correctement.*

Lancez le lecteur WAV pour tester l'application. Notez ceci :

- Le lecteur n'affiche pas de bouton Arrêt, sauf pendant la brève lecture du fichier. Le fichier s'arrête de lui-même. S'il était plus long, vous auriez la possibilité de l'arrêter.
- Le fichier est lu une fois, puis le lecteur se positionne à la fin du fichier. Vous pouvez cliquer sur le bouton Retour pour l'entendre de nouveau.
- Le bouton Enregistrer est actif. Vous pouvez enregistrer au début ou à la fin du fichier, puis rembobiner pour entendre le résultat.

Lecture de fichiers vidéo

Lire un fichier vidéo n'est très différent de lire un fichier audio. Le lecteur vidéo du contrôle multimédia étant un périphérique composite, vous devez impérativement fournir un nom de fichier. Vous devez également configurer le contrôle multimédia pour la lecture vidéo.

En fait, le contrôle multimédia exige un petit peu plus de vous quand il s'agit de lire des fichiers vidéos. En plus de la simple suite de boutons, il faut aussi un écran de projection pour la vidéo. Le contrôle le plus indiqué à cet effet est le contrôle PictureBox.

Vous avez, jusqu'ici, défini la propriété `DeviceType` du contrôle multimédia comme `CDAudio` pour lire les CD et comme `WaveAudio` pour lire les fichiers WAV, et cela dans la fenêtre Propriétés ou les Pages de propriétés. Mais si votre contrôle multimédia est appelé à lire plusieurs types de médias dans une même application, les différentes valeurs devront êtres définies en cours d'exécution à l'aide d'instructions d'affectation.

Votre application peut contenir plusieurs contrôles PictureBox. Le contrôle multimédia doit donc savoir à quel contrôle PictureBox envoyer la vidéo. C'est le rôle de la propriété `hWndDisplay`.

`hWnd` est un préfixe souvent employé en programmation Windows pour désigner un gestionnaire, ou plus précisément un contexte de périphérique. Les sorties Visual Basic ne s'affichent pas vraiment sur l'écran lui-même, mais dans des fenêtres. (En fait, on peut envoyer une sortie vers n'importe quel périphérique Windows à l'aide des mêmes commandes de bases, et les périphériques Windows sont plus virtuels que réels pour votre programme. Windows se charge de toutes les conversions compliquées pour envoyer la sortie vers une imprimante ou un écran couleur.) L'important est que votre programme n'envoie pas la sortie vidéo à l'écran, mais à un contexte de périphérique. Ce contexte de périphérique est presque toujours l'écran, mais le contrôle multimédia doit connaître le contexte de périphérique adéquat. Il peut ainsi déterminer vers quelle fenêtre la sortie vidéo doit être envoyée, et ajuster en conséquence la taille de cette fenêtre.

Assez parlé. Créez maintenant un nouveau projet. Nommez la feuille `frmVideo` et entrez `Lecteur vidéo` dans sa propriété `Caption`. Ajoutez un contrôle multimédia en haut de la feuille. Nommez ce contrôle `mmcVideo`.

Au centre de la feuille, ajoutez un contrôle PictureBox. Disposez-le tel que le montre la Figure 14.12. Nommez ce contrôle `picVideo`.

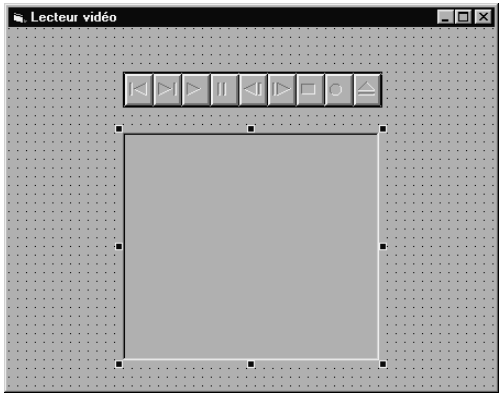


Figure 14.12
C'est dans le contrôle PictureBox que s'affichera la vidéo.

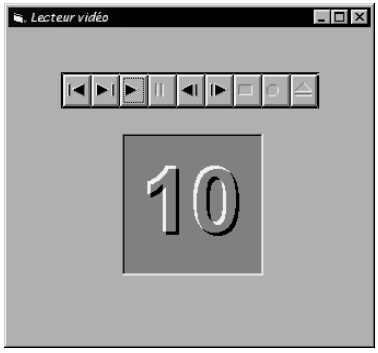
Définissez la propriété `DeviceType` comme `AVIVideo`. A la propriété `Filename`, affectez le fichier nommé Count24.AVI. Vous trouverez ce fichier dans le dossier \Graphics\Videos de Visual Basic. C'est presque terminé ! Créer un lecteur vidéo, vous le voyez, ce n'est pas sorcier. Maintenant, ajoutez la procédure événementielle `Form_Load()` du Listing 14.5.

Listing 14.5 : Associe le lecteur vidéo au contrôle PictureBox

```
1:  Private Sub Form_Load()
2:     ' Ouvre le lecteur vidéo.
3:     mmcVideo.Command = "Open"
4:     ' Connecte le lecteur vidéo au contrôle PictureBox.
5:     mmcVideo.hWndDisplay = picVideo.hWnd
6:  End Sub
```

Exécutez le programme. La Figure 14.13 montre un extrait du résultat.

Figure 14.13
Digne de Hollywood !



En travaillant avec le contrôle multimédia, vous découvrirez des raccourcis. Par exemple, il est inutile de spécifier le type de périphérique pour les périphériques multimédias composites, car Visual Basic peut le déduire lui-même de l'extension du fichier utilisé.

Par ailleurs, vous n'avez point remarqué que les labels d'état ne s'actualisent pas toujours à temps. En l'occurrence, le label n'a affiché "Lecture" qu'une fois la lecture terminée ! L'événement `StatusUpdate` a lieu toutes les quelques millisecondes. Pour que la mise à jour se fasse plus souvent, donc avec plus d'exactitude, affectez à la propriété `UpdateInterval` une valeur plus petite (selon la valeur par défaut, `1000`, l'état se met à jour toutes les secondes).

Ne définissez pas pour `UpdateInterval` *une valeur trop petite, car l'application contenant le contrôle multimédia consommerait trop de temps à mettre à jour le label, ce qui ralentirait le système. De tels ralentissements provoquent souvent quelques tressautements lors de la lecture des fichiers multimédias.*

En résumé

Ce chapitre vous a montré combien il était facile de relever vos applications d'une pointe de graphisme et d'un trait de multimédia. Les contrôles Line et Shape permettent de mettre en valeur les éléments de la feuille ou de séparer les contrôles.

Visual Basic propose également des méthodes de dessin qui vous permettent de tracer des lignes, des cadres ou des ellipses, ainsi que de dessiner pixel par pixel. Ces méthodes sont assez simples, mais elles vous laissent le plein contrôle sur vos dessins.

Les dernières sections de ce chapitre ont détaillé le fonctionnement du contrôle multimédia. Le contrôle multimédia est l'un des contrôles les plus complets de Visual Basic, car il supporte de multiples types de périphériques. Il permet à vos programmes de lire des CD audio, des fichiers son et même des vidéos. Le contrôle multimédia peut être piloté par l'utilisateur à l'aide de boutons, ou depuis le code à l'aide de la propriété `Command`.

Le chapitre suivant explique comment ajouter des modèles de feuilles à vos projets afin de standardiser vos applications.

Questions-réponses

Q Puis-je utiliser le contrôle multimédia pour afficher de simples images ?

R Le contrôle multimédia, comme son nom l'indique, est destiné aux périphériques multimédias. Ce qui n'inclut pas les simples images et photos. Pour les fichiers de ce type, optez plutôt pour les contrôles Image ou PictureBox.

Q Vaut-il mieux utiliser les contrôles de dessin ou les méthodes ?

R Les deux ne s'excluent pas. Les contrôles Line et Shape prennent diverses formes que vous pouvez placer sur la feuille lors de la création ou définir à l'exécution par des valeurs de propriétés. Les méthodes de dessin offrent plus de flexibilité, en permettant de dessiner point à point. Elles permettent également de tracer des lignes et des cadres.

Atelier

L'atelier propose une série de questions sous forme de quiz, grâce auxquelles vous affermirez votre compréhension des sujets traités dans le chapitre, et des exercices qui vous permettront de mettre en pratique ce que vous avez appris. Il convient de comprendre les réponses au quiz et aux exercices avant de passer au chapitre suivant. Vous trouverez ces réponses à l'Annexe A.

Quiz

1. Quel contrôle dessine des cercles sur la feuille ?
2. Quelle méthode dessine des carrés sur la feuille ?
3. Laquelle des méthodes suivantes permet de dessiner des cadres ?

   a. `PSet`

   b. `Line`

   c. `Box`

   d. `Draw`

4. Les contrôles Line et Shape génèrent des dessins qui supportent E/S propriétés, mais pas des événements. Vrai ou faux ?
5. Quelle option de la méthode `Line` applique une couleur de fond aux cadres ?
6. Pourquoi Visual Basic n'active-t-il pas tous les boutons lorsque vous placez le contrôle multimédia sur une feuille ?
7. A quoi sert la propriété `Mode` ?
8. Comment faire en sorte que l'état du périphérique du contrôle multimédia se mette à jour plus souvent ?
9. Pourquoi doit-on utiliser un contrôle PictureBox pour les vidéos ?
10. Comment le contrôle multimédia sait-il à quel contrôle PictureBox il doit envoyer la sortie vidéo ?

Exercices

1. Ajoutez à l'application reproduite en Figure 14.5 des zones de liste Couleur de remplissage et Couleur de bordure, de façon à pouvoir aussi modifier ces paramètres. La Figure 14.14 montre ce à quoi votre feuille devrait ressembler.

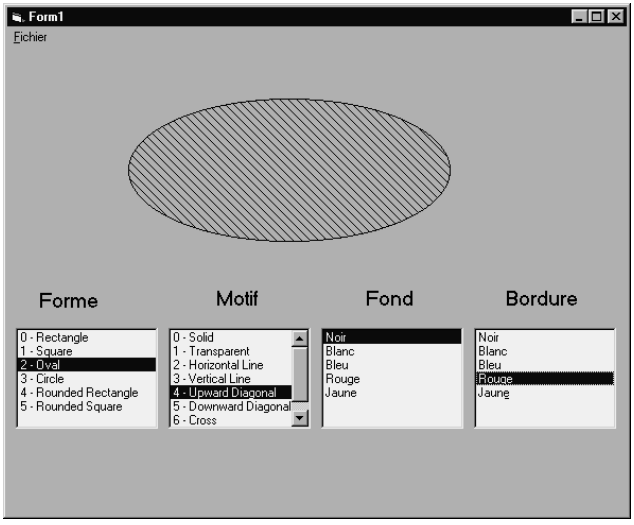


Figure 14.14
Vous avez maintenant plus de contrôle sur le dessin.

Truc. Utilisez une constante nommée, telle que `vbBlue`, pour répondre aux sélections de l'utilisateur. Vous n'êtes pas obligé de proposer toutes les couleurs dans toutes les zones de liste.

2. Exercez-vous au dessin de lignes et de cercles à l'aide des méthodes appropriées, mais sur un contrôle PictureBox cette fois. Le contrôle PictureBox accepte les mêmes méthodes de dessin que la feuille.
3. Reprenez le lecteur WAV que nous avons créé. Offrez à l'utilisateur la possibilité de choisir son fichier dans une boîte de dialogue Ouvrir.

PB7

Les barres de défilement

En plus de vous proposer un peu d'entraînement, ce Projet bonus introduit deux nouveaux contrôles : les barres de défilement verticale et horizontale. Ils sont toutefois très simples à manipuler, leur comportement étant en grande partie contrôlé par les propriétés définies lors de l'exécution. Il suffit ensuite d'écrire les procédures événementielles qui répondront aux actions de l'utilisateur.

Présentation des barres de défilement

Beaucoup des applications que vous avez déjà créées contenaient des barres de défilement, car certains contrôles les intègrent d'office. Par exemple, une zone de texte multiligne qui contient beaucoup de données affichera des barres de défilement, à moins que vous ne spécifiez le contraire dans la propriété `ScrollBars`. Sans les barres de défilement, l'utilisateur serait incapable d'accéder à la totalité du contenu. Les barres de défilement sont également intrinsèques aux zones de liste déroulante. La Figure PB7.1 détaille les différentes parties de chaque barre de défilement.

Les barres de défilement ne sont pas seulement utiles dans les zones de texte et zones de liste. Elles peuvent également offrir à l'utilisateur un contrôle flexible sur des valeurs graduelles, telles que la profondeur d'une couleur, la taille d'une image, ou autres valeurs numériques.

La Boîte à outils comporte les contrôles HScrollBar et VScrollBar, qui correspondent respectivement aux barres de défilement horizontale et verticale. Une fois ces contrôles disposés sur la feuille, vous devez régler les propriétés qui en définissent le comportement.

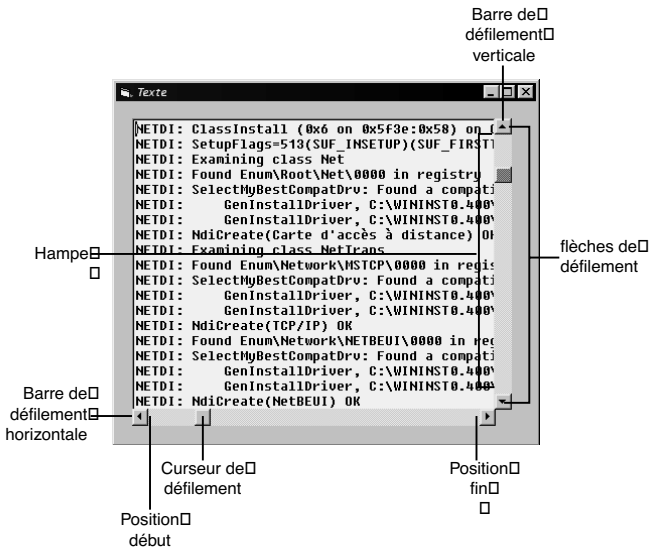


Figure PB7.1
Les barres de défilement horizontale et verticale peuvent apparaître automatiquement.

Fonctionnement des barres de défilement

Une fois que vous avez saisi le fonctionnement des barres de défilement, il ne vous reste pas grand-chose à apprendre pour être en mesure de les programmer. Voici les principes de ce fonctionnement :

- Lorsque le curseur de la barre de défilement horizontale est mis à la position la plus à gauche (le plus en haut pour la barre de défilement verticale), la valeur de défilement est la plus basse.
- Déplacé vers la droite ou la gauche (vers le haut ou le bas pour la barre de défilement verticale), le curseur décroît ou accroît d'autant la valeur de défilement correspondante.
- Lorsque le curseur de la barre de défilement horizontale est mis à la position la plus à droite (le plus en bas pour la barre de défilement verticale), la valeur de défilement est la plus haute.
- En cliquant sur les flèches de défilement, on décroît ou accroît de petites unités la valeur de défilement.
- En cliquant d'un côté ou de l'autre du curseur de défilement (sur la hampe, donc), on décroît ou accroît de grandes unités la valeur de défilement.

Propriétés des barres de défilement

Le Tableau PB7.1 présente les propriétés les plus importantes des barres de défilement. Ces propriétés peuvent être définies lors de l'exécution, mais on s'en occupe généralement pendant la phase de création. Ces propriétés correspondent aux principes de fonctionnement décrits ci-dessus.

Tableau PB7.1 : Propriétés des contrôles HScrollBar et VScrollBar

Propriété	*Description*
`LargeChange`	Spécifie la valeur selon laquelle la barre de défilement change le positionnement lorsqu'on clique sur la hampe (d'un côté ou de l'autre du curseur de défilement).
`Max`	Nombre maximal d'unités correspondant à la valeur de défilement la plus haute. La valeur par défaut est `32 767`.
`Min`	Nombre minimal d'unités correspondant à la valeur de défilement la plus basse. La valeur par défaut est `1`.
`SmallChange`	Spécifie la valeur selon laquelle la barre de défilement change le positionnement lorsqu'on clique sur les flèches de défilement.
`Value`	Spécifie l'unité de mesure courante pour la valeur de défilement.

La plage par défaut des valeurs de défilement va de `1` à `32 767`. Mais vous pouvez ajuster cete plage aux besoins de votre application. Supposons, par exemple, que vous écriviez un programme qui traite des montants allant de 3000 F à 60 000 F. Vous pourriez obtenir les montants de l'utilisateur par l'intermédiare de zones de texte. Mais il serait plus simple de fournir une espèce de gradateur, sans la forme d'une barre de défilement. Les valeurs de défilement seraient alors incluses dans les valeurs de propriétés suivantes :

```
Min: 3000
Max: 60000
```

Il suffit ensuite de définir les incréments correspondant aux valeurs de défilement. Supposons que les montants traités par votre programme puissent être incrémentés de 500 F. Le montant (indiqué dans la propriété `Value` de la barre de défilement) peut alors changer par tranches de 500 F si l'utilisateur clique sur les flèches (petit changement), ou par tranches de 2500 F (grand changement) s'il clique sur la hampe. Les propriétés devraient alors être réglées comme ceci :

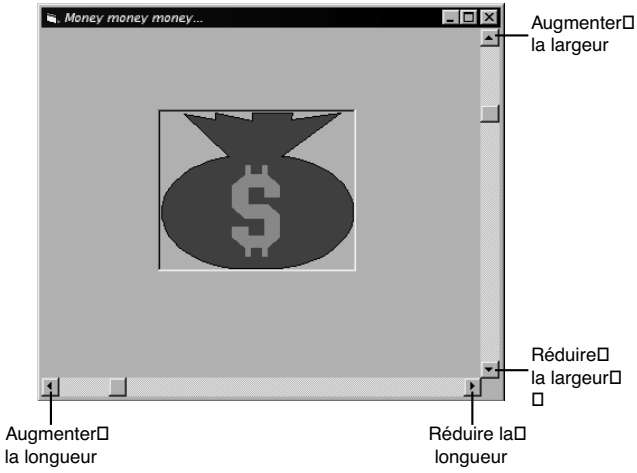
```
LargeChange: 2500
SmallChange: 500
```

Vous êtes libres de modifier la taille de la barre de défilement, en hauteur comme en largeur. La taille "physique" de la barre de défilement n'affecte en rien les valeurs de défilement.

Créer l'application

La Figure PB7.2 montre l'écran de l'application que vous allez maintenant créer. L'utilisateur se sert des barres de défilement pour modifier la taille de l'image. Le code requis est minime, parce que l'essentiel du travail est effectué par les propriétés des barres de défilement.

Figure PB7.2
Les barres de défilement contrôlent la taille de l'image.



Gardez à l'esprit le sens dans lequel les barres de défilement fonctionnent. Dans notre petite application, pour accroître la taille de l'image, il faut accroître la valeur de défilement, c'est-à-dire faire glisser le curseur de feuilles vers le bas. Intuitivement, on penserait plutôt réduire la taille de l'image en faisant descendre le curseur. Rien n'empêche d'écrire l'application de façon à inverser ces directions. Pour notre exemple, toutefois, nous nous en sommes tenus aux paramètres par défaut.

Cette application exploite les fichiers graphiques livrés avec Visual Basic. Si vous ne les avez pas installés, faites-le, copiez le fichier requis sur votre disque, ou travaillez directement sur le CD-ROM Visual Basic.

Le Tableau PB7.2 détaille les contrôles nécessaires à notre application.

Tableau PB7.2 : Contrôles et propriétés à configurer pour la feuille

Contrôle	*Propriété*	*Valeur*
Feuille	`Name`	`frmScoll`
Feuille	`Caption`	`Money money money`
Feuille	`Height`	`4650`
Feuille	`Width`	`5295`
PictureBox	`Name`	`picScroll`
PictureBox	`Height`	`1600`
PictureBox	`Left`	`1560`
PictureBox	`Picture`	`Common\Graphics\Metafile\Business \Moneybag`
PictureBox	`Top`	`1200`
PictureBox	`Width`	`1575`
Barre de défilement verticale	`Name`	`vscScroll`
Barre de défilement verticale	`Height`	`3975`
Barre de défilement verticale	`LargeChange`	`100`
Barre de défilement verticale	`Left`	`4920`
Barre de défilement verticale	`Max`	`3750`
Barre de défilement verticale	`Min`	`1600`
Barre de défilement verticale	`SmallChange`	`50`
Barre de défilement verticale	`Top`	`0`
Barre de défilement verticale	`Width`	`255`
Barre de défilement horizontale	`Name`	`hscScroll`
Barre de défilement horizontale	`Height`	`255`
Barre de défilement horizontale	`LargeChange`	`100`

Tableau PB7.2 : Contrôles et propriétés à configurer pour la feuille *(suite)*

Contrôle	*Propriété*	*Valeur*
Barre de défilement horizontale	`Left`	`0`
Barre de défilement horizontale	`Max`	`3750`
Barre de défilement horizontale	`Min`	`1600`
Barre de défilement horizontale	`SmallChange`	`50`
Barre de défilement horizontale	`Top`	`3960`
Barre de défilement horizontale	`Width`	`4935`

Chaque fois que l'utilisateur clique sur une flèche de défilement, la taille de l'image change de 50 pixels, tel que spécifié dans la propriété `SmallChange` des deux barres de défilement. Chaque fois que l'utilisateur clique sur la hampe, la taille de l'image change de 100 pixels, tel que spécifié dans la propriété `LargeChange`.

Ajouter le code

Le Listing PB7.1 fournit le code de ce projet.

La taille de l'image ne doit pas être seule à changer, mais également la position : l'image doit rester centrée.

Listing PB7.1 : Les barres de défilement modifient la taille de l'image

```
1:  Private Sub hscScroll_Change()
2:     ' Change la largeur et la position horizontale de l'image.
3:     picScroll.Width = hscScroll.Value
4:     picScroll.Left = (frmScroll.Width / 2) - (picScroll.Width
   ➥/ 2) - 300
5:  End Sub
6:
7:  Private Sub vscScroll_Change()
8:     ' Change la hauteur et la position verticale de l'image.
9:     picScroll.Height = vscScroll.Value
10:    picScroll.Top = (frmScroll.Height / 2) - (picScroll.Height
   ➥/ 2) -300
11: End Sub
12:
13: Private Sub vscScroll_Scroll()
```

```
14:   ' Répond au curseur de défilement.
15:   Call vscScroll_Change
16: End Sub
17:
18: Private Sub hscScroll_Scroll()
19:   ' Répond au curseur de défilement.
20:  Call hscScroll_Change
21: End Sub
```

Analyse

Le code est répétitif, parce que les deux barres de défilement font foncièrement la même chose : contrôler les propriétés `Width` et `Height` de l'image. A la ligne 3, la première procédure modifie la largeur de l'image en fonction de la valeur de la barre de défilement horizontale. (Pour les deux barres de défilement, les propriétés `Min` et `Max` ont été définies, lors de la création, de façon que la taille de l'image garde toujours une taille raisonnable.) La ligne 4 spécifie la nouvelle position de l'image redimensionnée. Cette position est fixée en fonction de la hauteur de l'image par rapport à la hauteur de la feuille. `frmScroll .Width / 2` définit le milieu de la feuille, `picScroll.Width / 2` celui de l'image. Les 300 pixels soustraits correspondent à la largeur de la barre de défilement verticale.

Les lignes 7 à 11 font exactement la même chose, mais en fonction de la barre de défilement horizontale et pour la largeur de l'image.

Les deux dernières procédures ont cela de particulier qu'elles ne font qu'appeler les autres procédures. La première de ces deux procédures s'exécute quand l'utilisateur clique sur l'une des flèches ou sur la hampe. Mais qu'en est-il du curseur de défilement ? Sans les procédures des événements `Change`, l'image ne bougerait pas tant que l'utilisateur n'a pas relâché le bouton de la souris — c'est-à-dire tant que l'événement `Scroll` n'est pas achevé. En appelant la procédure événementielle `Scroll` depuis la procédure événementielle `Change`, on synchronise les mouvements du curseur de feuille aux modifications de la taille. En fait, la procédure événementielle `Scroll` est appelée pendant tout le déplacement du curseur, pour chacune des valeurs par lesquelles il passe.

On aurait pu placer le code de la procédure événementielle `Click` directement dans la procédure événementielle `Change`, pour ne garder que cette dernière. La procédure événementielle `Change` s'exécuterait chaque fois que l'utilisateur agit sur une barres de défilement, le curseur, les flèches et la hampe pouvant tous générer des événements `Change`. On aurait pu, certes, mais vous n'avez pas encore appris à appeler une procédure depuis une autre, et cet exemple s'y prêtait parfaitement.

Résumé de la Partie II

Vous êtes maintenant passé du stade de débutant à celui de programmeur avancé. L'environnement de programmation Visual Basic vous est tout à fait familier, et vous maîtrisez l'essentiel du langage.

Dans les chapitres qui viennent, il s'agira donc surtout d'affûter et de perfectionner les notions et les techniques déjà acquises. Vous avez maintenant un bagage suffisant pour rechercher par vous-même, en dehors de cet ouvrage, des informations complémentaires. Manuels plus avancés, revues et sites Web spécialisés vous permettront de perfectionner votre pratique de Visual Basic et de devenir rapidement un authentique développeur professionnel.

Voici ce que nous avons étudié dans ces sept chapitres :

- **La structure des applications.** Un programme Visual Basic est plus qu'un simple assemblage de contrôles et de procédures événementielles. L'application peut être constituée de plusieurs feuilles et modules de code travaillant en commun (Chapitre 8).
- **La portée des variables.** Alors que vos codes deviennent plus modulaires (et donc plus faciles à maintenir), vous devez toujours garder à l'esprit la portée des variables, qui détermine quelles procédures auront accès à quelles variables de l'application (Chapitre 8).
- **Passer des données.** Qu'il s'agisse de sous-routines ou de fonctions, vos procédures doivent être en mesure de passer des données aux autres procédures (Chapitre 8).
- **Passer des contrôles.** Les procédures peuvent, entre elles, se passer tous types d'objets : variables, mais aussi contrôles (Chapitre 8).
- **Les fonctions internes.** Grâce à une exploitation adéquate des fonctions internes, vous allégerez votre travail de programmation pour ce qui est des tâches communes, telles que calculer une racine carrée ou convertir une chaîne (Chapitre 8).

- **Les boîtes de dialogues communes.** Le contrôle Common Dialog permet d'offrir aux utilisateurs une interface standard pour l'ouverture et l'enregistrement de fichiers, la sélection de couleurs ou l'impression (Chapitre 9).
- **La gestion de la souris.** Vos programmes peuvent suivre les déplacements et les clics de la souris (Chapitre 10).
- **Les contrôles de listes.** Visual Basic met à votre disposition divers types de zones de liste (Chapitre 10).
- **Le contrôle Timer.** Le contrôle Timer interroge l'horloge interne du PC pour exécuter des procédures à intervalles précis (Chapitre 10).
- **Les tableaux.** Les simples variables ne peuvent stocker des listes de données. Pour traiter de grandes quantités de données, il faut les stocker dans des tableaux de variables (Chapitre 10).
- **Les feuilles.** Comme les autres objets Visual Basic, les feuilles supportent une vaste gamme de propriétés, événements et méthodes, permettant notamment de les masquer et de les afficher à volonté lors de l'exécution (Chapitre 11).
- **Envoyer du texte aux feuilles.** La méthode `Print` permet d'envoyer du texte directement sur la feuille, sans passer par des contrôles (Chapitre 11).
- **Les barres d'outils.** Les barres d'outils sont un aspect capital de l'interface utilisateur (Chapitre 11).
- **Les coolbars.** Les coolbars sont relativement nouvelles dans les logiciels Windows, mais elles offent un moyen supplémentaire de dynamiser et personnaliser vos applications (Chapitre 11).
- **Le traitement des fichiers.** Le traitement des fichiers n'a rien de sorcier, mais réclame un peu de programmation (Chapitre 11).
- **Les fichiers séquentiels.** Les fichiers séquentiels se lisent et s'écrivent dans un ordre précis et immuable, mais sont faciles à manipuler (Chapitre 12).
- **Les fichiers aléatoires.** L'accès aléatoire permet de lire et d'écrire à n'importe quel endroit du fichier (Chapitre 12).
- **Les contrôles de fichiers.** Les contrôles ne manipulent pas eux-mêmes les fichiers, mais permettent à l'utilisateur de faire son choix parmi les lecteurs, dossiers et types de fichiers de son système (Chapitre 12).
- **L'impression.** A l'aide de méthodes spécifiques, votre application peut envoyer des sorties aux diverses imprimantes présentes sur le système de l'utilisateur (Chapitre 13).

- **L'impression de feuilles.** Vous pouvez, aussi bien, imprimer les feuilles de votre projet (Chapitre 13).
- **Les fichiers graphiques.** Les contrôles Image et PictureBox permettent d'afficher les images contenues dans des fichiers graphiques (Chapitre 14).
- **Les contrôles de dessin.** Grâce aux contrôles Line et Shape, vous pouvez enjoliver ou mettre en relief les éléments de la feuille (Chapitre 14).
- **Les méthodes de dessin.** Les méthodes de dessin permettent de tracer des images sur les feuilles comme sur le contrôle PictureBox (Chapitre 14).
- **Le contrôle multimédia.** Le contrôle multimédia permet de gérer, à l'intérieur de l'application, n'importe quel périphérique multimédia : lecteur CD, afficheur vidéo, etc. (Chapitre 14).

Partie III

D'un coup d'œil

Vous avez suivi un long trajet depuis que ce livre a commencé à vous enseigner comment écrire des programmes Visual Basic. Vous devez désormais vous considérer comme étant meilleur qu'un programmeur débutant. Vous pouvez déjà presque réaliser ce que vous voulez avec Visual Basic. Il vous reste maintenant à améliorer vos compétences.

La troisième et dernière partie de ce didacticiel commence par examiner certains aspects avancés de la programmation Visual Basic, ce qui ne signifie cependant pas "difficile". Presque tous les éléments que vous apprendrez dans cette partie sont considérés comme avancés. Quoi qu'il en soit, vous avez des bases si solides désormais que vous devriez acquérir assez facilement ces compétences supplémentaires.

Lorsque vous commencerez à explorer les objets et les contrôles ActiveX, vous verrez que ces éléments vous aident à produire un meilleur code et à écrire des programmes plus rapidement. Vous apprendrez à réutiliser les composants que vous créez de la même manière que vous avez réutilisé les procédures dans la Partie II. Vous développerez vos propres outils dans la fenêtre Boîte à outils de Visual Basic en créant vos propres contrôles ActiveX.

Comme vous le verrez, Visual Basic peut vous proposer une grande diversité de contrôles. Dans cette dernière partie, vous apprendrez comment accéder aux fichiers, gérer les données dans les formats courants de bases de données, et comment accéder à l'Internet en utilisant les outils fournis avec Visual Basic. Vous n'écrirez que très rarement une application qui exige plus de contrôles que ceux dont vous disposez déjà. Toutefois, vous serez soulagé d'apprendre que vous pourrez en obtenir d'autres, et même écrire les vôtres, si l'envie vous en prend. Vous pourrez utiliser ces contrôles non seulement dans l'environnement de programmation Visual Basic, mais également d'autres manières, par exemple les incorporer dans les pages Web pour les navigateurs compatibles avec ActiveX.

Une fois l'application écrite, il faut la tester à l'aide des outils de test et de débogage de Visual Basic. Si certains bogues sont difficiles à pister, Visual Basic va cependant assez loin pour vous aider à les localiser. Une fois votre application développée, vous devez la compiler pour la distribuer aux autres. A la fin de cette partie, vous apprendrez à empaqueter vos applications avec des routines d'installation pour permettre aux utilisateurs de les installer et de les exécuter.

Chapitre 15

Les modèles de feuilles

La leçon d'aujourd'hui vous montre les bénéfices que vous pouvez tirer des modèles de feuilles afin de donner une apparence plus uniforme aux feuilles créées dans un but commun. Visual Basic fournit plusieurs modèles de feuilles que vous pouvez ajouter, personnaliser et utiliser dans vos projets. Beaucoup de feuilles utilisées actuellement — comme la boîte de dialogue "A propos de" s'affichant lorsqu'on sélectionne Aide, A propos de — ont élaboré une forme de standard de facto dans la présentation, dont vous pouvez aussi bien suivre la tendance. La boîte de dialogue "A propos de" et quelques autres feuilles standards sont fournies avec Visual Basic.

Une fois que vous saurez utiliser les modèles de feuilles, vous pourrez créer les vôtres. Supposons que votre entreprise préfère que l'apparence de chaque boîte de dialogue soit normalisée et présente le nom de l'entreprise, le logo, l'heure et la date en haut de la feuille. En créant un modèle de feuille contenant ces informations, toutes les feuilles utilisées ensuite dans les applications pourront présenter ces éléments sans que vous ayez à les ajouter chaque fois que vous créez une nouvelle feuille dans un projet.

Vous apprendrez aujourd'hui :

- le rôle des modèles de feuilles ;
- pourquoi les utiliser ;
- comment les implémenter dans vos projets ;
- comment les ajouter à partir de l'assistant Création d'applications ;
- comment créer un fichier "Astuce du jour" ;
- comment ajouter vos propres modèles de feuilles à l'ensemble fourni avec Visual Basic.

A propos des modèles de feuilles

Un modèle de feuille est une sorte de gabarit. Lorsque vous démarrez avec un modèle de feuille, au lieu de partir d'une fenêtre vierge, vous économisez du temps, et vos feuilles ont une apparence uniforme.

Supposons que vous vouliez ajouter un écran Astuce du jour à votre application pour permettre aux utilisateurs de lire une astuce différente chaque fois qu'ils démarrent le programme. L'Astuce du jour la plus connue est sans nul doute celle qui s'affiche au démarrage de Windows 95. D'autres applications célèbres utilisent aussi un écran Astuce du jour, qui se présente souvent comme celui de Windows 95. La plupart ont par exemple une case à cocher en bas de la fenêtre qui permet d'empêcher qu'elle se réaffiche lors des démarrages suivants de l'application.

Comme vous l'avez déjà appris dans ce livre, la création d'applications normalisées présente de nombreux avantages : les utilisateurs s'y adaptent plus rapidement, ils seront plus à même de les apprécier, ils seront enclins à utiliser les mises à jour, ils apprendront plus vite à les utiliser et vous aurez moins d'appels de support. Si ces raisons ne suffisent pas à vous inciter à utiliser des écrans et des menus standards, vous verrez aussi que vous gagnerez du temps : votre application sera terminée plus vite et vous aurez moins de bogues à supprimer.

La Figure 15.1 montre le modèle de feuille Astuce du jour fournie par Visual Basic. Si vous aviez à ajouter un écran Astuce du jour à une application, préféreriez-vous l'utiliser ou partir d'une fenêtre vierge ? Vous choisirez évidemment le modèle. Il donne une apparence uniforme et demande moins de travail.

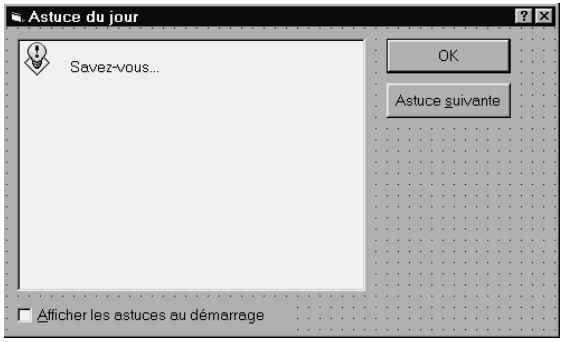


Figure 15.1
Visual Basic propose ce modèle de feuille Astuce du jour.

Info

Comme pour les squelettes d'applications créées par l'assistant Création d'applications, un modèle de feuille contient des emplacements pour les éléments courants, mais vous aurez à le modifier. Vous devrez remplacer

les éléments inutiles et personnaliser la feuille pour l'adapter aux exigences de votre application. Malgré cela, vous achèverez la feuille bien plus rapidement qu'en partant d'une feuille vierge.

Les modèles de feuilles contiennent des éléments visuels, tels que des icônes et des contrôles, et le code sous-jacent pour vous aider à l'intégrer dans votre application.

Un modèle de feuille n'est rien d'autre qu'une feuille avec des éléments déjà placés. Ce n'est pas un type de feuille particulier, mais plutôt un ensemble de feuilles prédéfinies. Lorsque vous créez des feuilles que vous souhaitez réutiliser, vous pouvez tout simplement les ajouter à la collection des modèles de votre système. Nous l'expliquerons ci-dessous.

Les modèles de feuilles proposés

Visual Basic fournit les modèles de feuilles suivants :

- **A propos de**. Généralement affiché par l'option de menu Aide, A propos de.
- **Navigateur**. Pour une simple navigation sur le Web.
- **Feuille de données**. Pour gérer les tables de type base de données.
- **Boîte de dialogue**. Pour créer des boîtes de dialogue.
- **Boîte de dialogue Connexion**. Pour demander un nom et un mot de passe.
- **Boîte de dialogue Connexion ODBC**. Pour les activités liées à ODBC.
- **Boîte de dialogue Option**. Pour gérer les boîtes de dialogues multipages et les paramétrages de personnalisation.
- **Ecran d'accueil**. Pour afficher un écran de démarrage au cours du chargement de l'application, qui n'apparaît habituellement que quelques secondes.
- **Astuce du jour**. Pour ajouter une astuce au démarrage.

ODBC (Open Database Connectivity) signifie connexion ouverte aux bases de données, et fournit un ensemble de commandes standard pour accéder à différents types de données enregistrés sur divers types d'ordinateurs.

Le reste de cette leçon décrit comment mettre en place nombre de ces modèles de feuilles dans vos applications. Certains sont trop particuliers pour entrer dans le cadre de ce chapitre : le Chapitre 19 traite plus en détail de la navigation sur l'Internet en utilisant le modèle Navigateur ; le Chapitre 18 décrit

comment lier les applications Visual Basic aux informations de base de données, à l'aide des modèles Feuilles de données ou ODBC.

Les sections suivantes passent en revue les modèles de feuilles les plus courants que vous utiliserez. Vous apprendrez à utiliser l'assistant Création d'applications pour ajouter des modèles et vous verrez en détail comment ajouter et personnaliser la boîte de dialogue A propos de dans vos applications. Les autres modèles n'en diffèrent que par les options qu'ils présentent.

Voici les étapes générales pour ajouter un modèle de feuille à votre application :

1. Ajouter le modèle de feuille dans la fenêtre Projet de l'application ;
2. Le personnaliser avec les détails exigés par votre application ;
3. Connecter le modèle à votre projet par programmation.

L'assistant Création d'applications

Vous pouvez ajouter un modèle de feuille à votre application à tout moment. Si vous utilisez l'assistant Création d'applications, vous devez cependant lui signaler que vous souhaitez utiliser un modèle de feuille particulier afin qu'il s'occupe de tous les détails et l'ajoute automatiquement.

La Figure 15.2 montre la boîte de dialogue de l'assistant Création d'applications dans laquelle vous spécifiez un modèle de feuille. Comme vous pouvez le voir, l'assistant ne propose que quatre options : Ecran de présentation au démarrage, Boîte de dialogue de connexion, Boîte de dialogue de paramétrage d'options et Boîte de dialogue A propos de.

Vous pouvez cependant ajouter d'autres modèles en cliquant sur le bouton Modèles de feuilles.

La liste qui s'affiche comporte non seulement les feuilles fournies avec Visual Basic, mais aussi celles que vous avez ajoutées à la bibliothèque des modèles.

Ajouter des modèles de feuilles à une application

L'ajout d'un modèle de feuille à une application s'effectue, comme pour toute autre feuille, de l'une des deux manières suivantes :

- Sélectionnez l'option Ajouter une feuille dans le menu Projet et choisissez un modèle dans la liste des icônes de modèles qui s'affiche dans la boîte de dialogue, illustrée à la Figure 15.3.

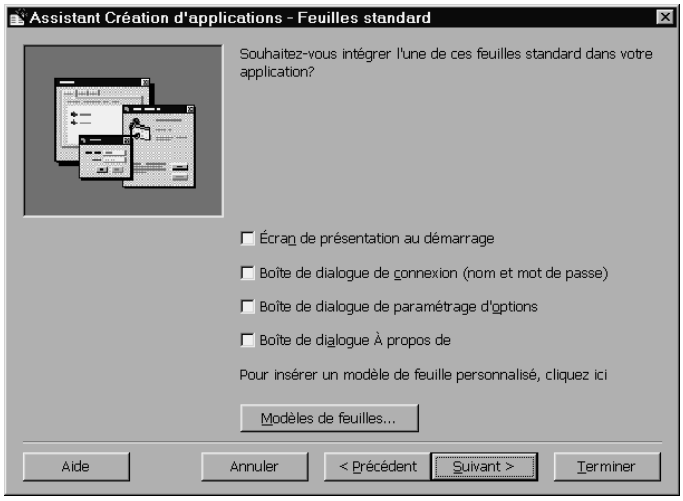


Figure 15.2
L'assistant Création d'applications ajoutera des feuilles à votre application.

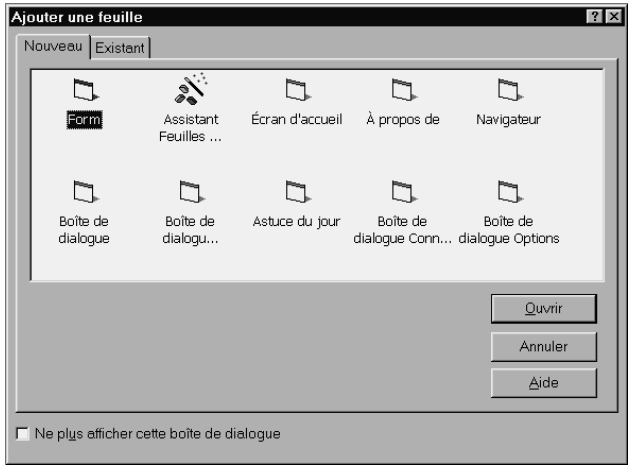


Figure 15.3
L'ajout d'un modèle de feuille est simple.

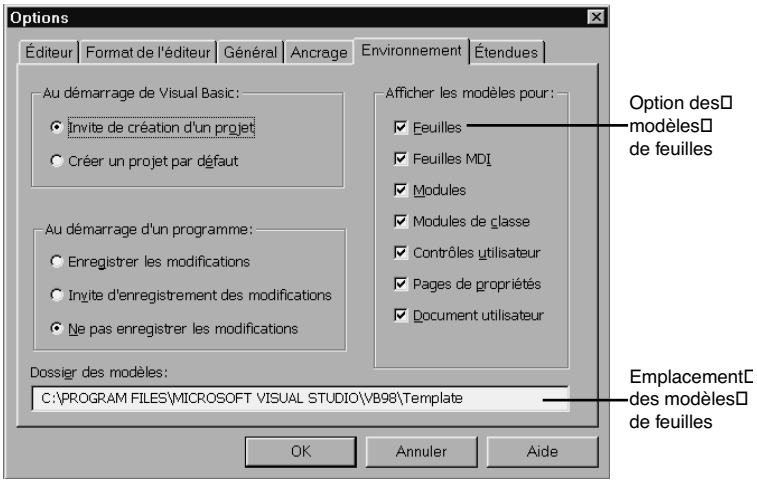
- Cliquez du bouton droit dans la fenêtre Projet, sélectionnez Ajouter, Feuille et sélectionnez le modèle dans la liste.

Pour ajouter une feuille vierge, sélectionnez la première icône de la liste (intitulée Form). Pour ajouter un modèle, sélectionnez l'icône correspondante.

Modifier les modèles

Lorsque vous sélectionnez Outils, Options et que vous cliquez sur l'onglet Environnement, Visual Basic affiche la boîte de dialogue illustrée à la Figure 15.4. En cochant la case Feuilles, vous indiquez à Visual Basic s'il doit proposer la liste des modèles de feuilles dans l'assistant Création d'applications et dans la boîte de dialogue Projet, Ajouter une feuille.

Figure 15.4
Vous pouvez contrôler les modèles proposés par Visual Basic.



Comme le montre cette boîte de dialogue, Visual Basic fournit des modèles pour bien d'autres objets que les feuilles. Il existe des modèles de modules, de contrôles, de pages de propriétés, etc. Si certains, ne correspondant pas à des feuilles, ne sont utilisés que dans des applications assez avancées, vous pouvez constater qu'il existe un modèle pour presque tout type d'objet que vous pourrez créer.

Si vous ne cochez pas l'option Feuille pour masquer l'affichage des modèles, lorsque vous créez un nouveau projet, Visual Basic crée automatiquement une feuille vierge dans votre projet sans vous laisser la possibilité de sélectionner un autre type de feuille.

La boîte de dialogue Options montre le dossier où Visual Basic recherche les modèles de feuilles. Il s'attend à ce que tous les modèles se trouvent dans ce dossier. Si vous configurez diverses bibliothèques de modèles pour des usages divers (ce qu'un programmeur indépendant pourrait faire s'il travaille avec plusieurs entreprises), vous

pouvez enregistrer chacun des ensembles de modèles dans des dossiers différents. Quand vous souhaitez travailler avec un de ces ensembles, entrez le chemin dans la boîte de dialogue Options.

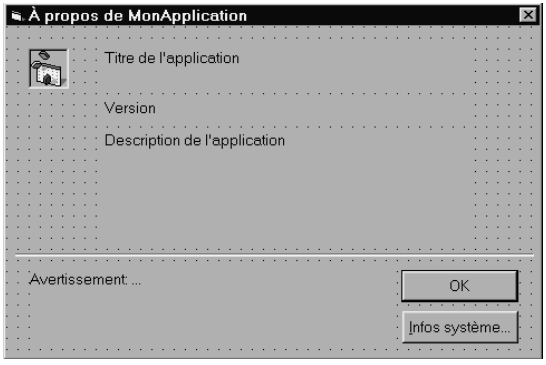
Faire
Pour modifier de manière permanente un modèle et l'adapter à vos besoins si vous utilisez souvent le même, enregistrez-en une copie sous un autre nom et modifiez cette dernière. Utilisez ensuite la copie.

Ne pas faire
Ne faites pas de modifications directement sur le modèle d'origine.

Le modèle de feuille A propos de

Le but de chaque modèle de feuille est différent, mais la procédure générale de connexion du modèle à une application est toujours la même. Dans cette section, vous allez ajouter un modèle de feuille à une application pour vous familiariser avec le processus. Une des boîtes de dialogue les plus courantes, qui s'affiche dans la plupart des applications Windows, est la boîte de dialogue A propos de, lorsqu'on sélectionne l'option de menu Aide, A propos de. La Figure 15.5 montre à quoi ressemble le modèle dans la fenêtre de l'environnement Visual Basic.

Figure 15.5
Le modèle de feuille de la boîte de dialogue A propos de, qui s'affiche dans la plupart des applications Windows.



Pour vous donner une idée de la différence entre le modèle et la feuille à laquelle vous allez aboutir, voyez la Figure 15.6. Elle montre la boîte de dialogue A propos de Visual Basic. Comme vous le constatez, le modèle fournit les emplacements qui contiendront les informations que vous y placerez dans l'application finale.

Figure 15.6
La boîte de dialogue A propos de Visual Basic correspond au format du modèle de feuille.

Pour mettre en pratique l'ajout d'un modèle de feuille A propos de, démarrez une nouvelle application et suivez ces étapes :

1. Ouvrez le Créateur de menus, ajoutez Aide à la barre de menus, indiquez A propos de comme unique option du menu Aide et nommez-la `mnuHelpAbout`.
2. Nommez la feuille du projet `frmTestAbout` et changez son titre en Démonstration de la boîte de dialogue A propos de.
3. A partir du menu Fichier, choisissez Enregistrer la feuille sous et tapez `Feuille A propos de` comme nom de fichier.
4. Choisissez Enregistrer le projet sous dans le menu Fichier et tapez `Projet A propos de` comme nom de projet (la boîte de dialogue que vous ajouterez utilisera le nom de projet).
5. Dans le menu Projet, sélectionnez Ajouter une feuille pour afficher la boîte de dialogue correspondante. Double-cliquez sur A propos de pour ajouter la boîte de dialogue, qui n'est rien d'autre qu'une feuille avec des contrôles. Elle est ajoutée à votre application sous le nom `frmAbout`.
6. Utilisez l'option du menu Fenêtre pour revenir à votre feuille de départ (`frmTestAbout`). Ajoutez la procédure événementielle suivante :

```
Private Sub mnuHelpAbout_Click()
  frmAbout.Show
End Sub
```

7. Vous pouvez maintenant exécuter l'application. Lorsque vous choisissez A propos de dans le menu Aide, la boîte de dialogue s'affiche. Le modèle de feuille A propos de connaît le nom du projet et l'affiche dans la zone de titre.
8. Cliquez sur OK pour fermer la boîte de dialogue A propos de, puis fermez la feuille principale pour arrêter l'application.

Le modèle de la boîte de dialogue A propos de connaît le nom du projet et l'affiche dans la zone de titre. Le nom provient de l'objet `App`, que nous étudierons dans la leçon de demain.

Utilisez le Menu Fenêtre pour afficher la feuille A propos de dans la zone d'édition. La procédure événementielle `Form_Load()` de la boîte de dialogue contient le code suivant qui initialise le titre à partir de l'objet `App` :

```
1: Private Sub Form_Load()
2:     Me.Caption = "À propos de " & App.Title
3:     lblVersion.Caption = "Version " & App.Major & "." &
       ➥App.Minor & "." & App.Revision
4:     lblTitle.Caption = App.Title
5: End Sub
```

La ligne 3 configure les numéros de version majeur, mineur et de révision de l'application dans la procédure événementielle `Form_Load()` de la feuille principale. Vous pouvez supprimer les références à ces valeurs du module de la feuille A propos de (et les étiquettes les affichant) si vous ne souhaitez pas les utiliser.

Le module de la boîte de dialogue A propos de contient du code initialisant automatiquement les valeurs titre et numéro de version, mais pas certaines étiquettes telles que la description de l'application et la zone d'avertissement. Vous devez initialiser la propriété `Caption` de la description (étiquette `lblDescription`). Si vous ne souhaitez pas de zone d'avertissement ou de copyright, supprimez l'étiquette correspondante (`lblDisclaimer`). Vous pouvez insérer à la place une icône ou une image pour attirer l'attention de l'utilisateur.

La boîte de dialogue A propos de implique plus de code que le simple affichage du nom et du numéro de version de l'application, par exemple le code permettant de fermer la fenêtre lorsque l'utilisateur clique sur OK. Mais la véritable puissance de la boîte de dialogue A propos de réside dans la procédure événementielle du bouton de commande Infos système :

```
1:  Private Sub cmdSysInfo_Click()
2:    Call StartSysInfo
3:  End Sub
```

`StartSysInfo` est une procédure générale listée un peu plus loin dans le module de code de la boîte de dialogue A propos de. Elle lance un programme système nommé MSINFO32.EXE qui se trouve dans un dossier de Windows. Vous pourriez remplacer ce programme par votre propre code, mais pourquoi ne pas se tenir au programme standard d'informations sur le système que les utilisateurs sont habitués à voir dans les autres applications Windows ?

Si vous exécutez une nouvelle fois votre application, affichez la boîte de dialogue A propos de et cliquez sur le bouton Infos système, l'application correspondante démarre. (On dit que c'est un *processus enfant* de votre application.) La Figure 15.7 montre la fenêtre Infos système.

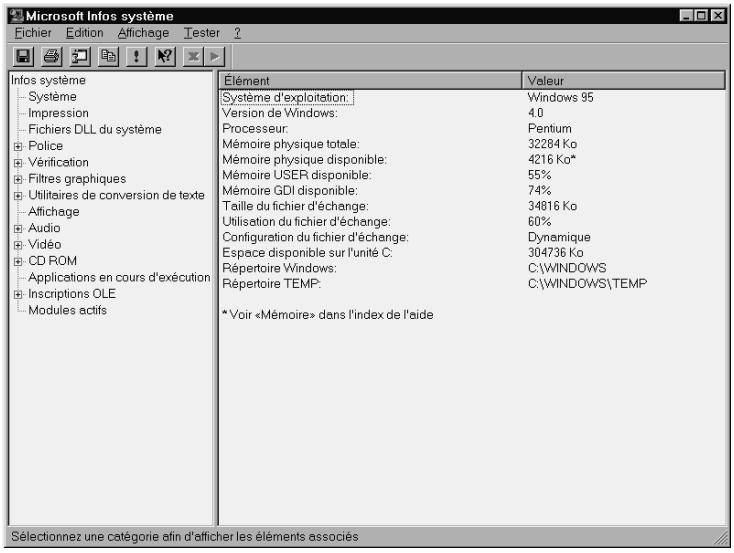


Figure 15.7
Le code accompagnant la boîte de dialogue A propos de affiche la fenêtre Infos système.

La fenêtre Infos système peut différer de celle illustrée à la Figure 15.7, suivant la configuration de votre machine.

Autres modèles de feuilles

Maintenant que vous avez créé un projet utilisant la boîte de dialogue A propos de, vous n'aurez pas de problème à placer les trois autres modèles principaux de feuilles. Quand vous sélectionnez Ajouter une feuille dans le menu Projet, vous avez la possibilité d'ajouter plusieurs modèles au projet en cours.

Cette section étudie plus en détail les modèles de feuilles suivants :

- l'écran d'accueil ;
- la boîte de dialogue Connexion ;
- la boîte de dialogue Astuce du jour ;
- la boîte de dialogue Connexion ODBC.

Les applications ne doivent pas toujours inclure tous les modèles. La boîte de dialogue A propos de faisant partie de la plupart des applications Windows, il est bon de prendre l'habitude de l'inclure systématiquement. L'utilisation des autres modèles dépend surtout du but et des exigences de l'application. Cette section les étudie en détail, pour vous permettre de connaître ceux que vous voudrez ajouter à vos projets.

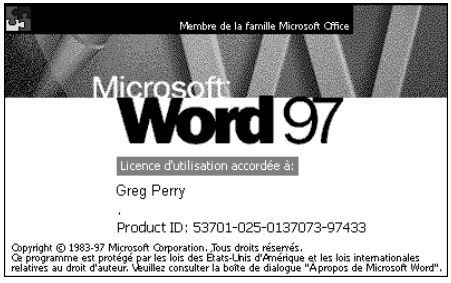
Dans cette section, vous apprendrez comment connecter les différents modèles à vos applications. Si vous les ajoutez au moment où vous utilisez l'assistant Création d'applications, vous vous épargnerez une partie du travail pris en charge par l'assistant. Il sera ainsi plus facile d'insérer les modèles dans vos projets.

Ecran d'accueil

Un *écran d'accueil* affiche un message d'introduction et éventuellement des informations de copyright et de contact sur le projet. (Bien qu'on parle d'*écran*, c'est en fait une feuille de la collection `Forms` de votre projet). Son objet premier est de saluer l'utilisateur. Contrairement à la boîte de dialogue A propos de, l'écran d'accueil ne s'affiche qu'au démarrage de l'application.

Un écran d'accueil affiche souvent une image accompagnée d'un écran de présentation. La Figure 15.8 montre l'écran d'accueil affiché au démarrage de Microsoft Word. Il comporte une image pour attirer l'attention et des informations sur le produit.

Figure 15.8
Un écran d'accueil identique à celui-ci apparaît lorsque l'application Word se charge.



Un écran d'accueil disparaît généralement au bout d'un court moment. Vous pouvez ajouter un bouton de commande ou du code qui vérifie une action clavier pour permettre à l'utilisateur de s'en débarrasser à sa convenance. Mais on utilise généralement un contrôle Timer pour afficher l'écran d'accueil pendant une durée déterminée. L'écran d'accueil sert aussi à masquer l'attente au démarrage due à l'initialisation des fichiers et des données.

L'écran d'accueil impose une contrainte absente de la boîte de dialogue A propos de. Vous devez indiquer à votre application de l'afficher avant la fenêtre normale. Il doit être configuré en tant que fenêtre de démarrage dans la boîte de dialogue Propriétés, à laquelle on accède par l'option Propriétés du menu Projet. Dès que le modèle Ecran d'accueil est ajouté au projet, la boîte de dialogue contient la feuille dans la liste des objets de démarrage. Vous devez aussi ajouter un bouton de commande ou un Timer à l'écran d'accueil pour afficher la fenêtre suivante quand il est temps de le faire.

Suivez ces étapes pour mettre en pratique la création d'un projet contenant un écran d'accueil :

1. Créez un nouveau projet. Vous laisserez telle quelle la feuille `Form1`.
2. Dans le menu Projet, choisissez Ajouter une feuille, puis sélectionnez l'écran d'accueil. Visual Basic affiche l'écran échantillon.
3. Modifiez les étiquettes pour qu'elles correspondent à celles de la Figure 15.9.

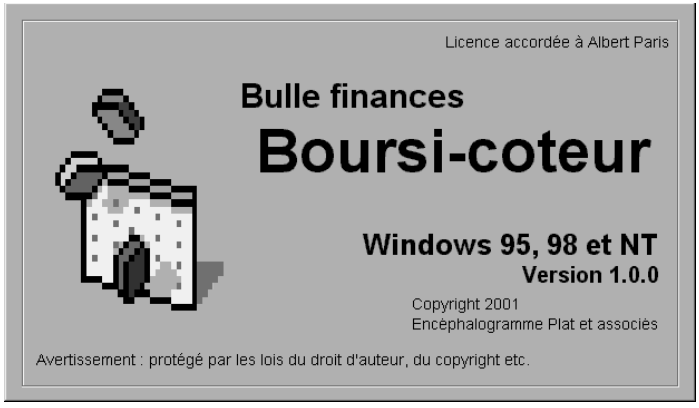


Figure 15.9
L'écran d'accueil est modifié.

4. Choisissez Propriétés dans le menu Projet. Indiquez `frmSplash` (c'est le nom de l'écran d'accueil) comme Objet de démarrage et cliquez sur OK.

5. Ajoutez la ligne suivante dans les événements `Form_Keypress()` et `Frame1 _Click()` (à la suite de l'instruction `Unload Me` des deux procédures) :

```
Form1.show 'Affiche la feuille normale
```

6. Exécutez l'application. La première feuille à s'afficher est l'écran d'accueil. Si vous appuyez sur une touche ou cliquez, il s'efface et laisse la place à la feuille normale, `Form1`. Fermez la feuille pour revenir dans l'environnement de développement de Visual Basic.

L'écran d'accueil prend automatiquement le nom de programme du projet, quel que soit le texte que vous saisissez dans l'étiquette centrale. Vous devez donc enregistrer le projet sous le nom Boursi-coteur ou modifier le code pour conserver l'étiquette d'origine.

Boîte de dialogue Connexion

Le besoin de sécurité s'accroît avec le nombre d'ordinateurs connectés. La boîte de dialogue Connexion est un modèle intéressant à ajouter à vos projets. Elle demande la saisie d'un nom d'utilisateur et d'un mot de passe et renvoie les valeurs à l'application pour traitement. La Figure 15.10 montre la boîte de dialogue Connexion.

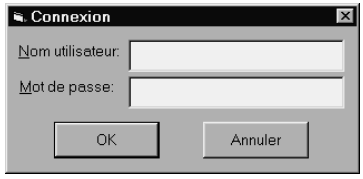


Figure 15.10
Utilisez la boîte de dialogue Connexion pour demander un nom d'utilisateur et un mot de passe.

Quand un utilisateur entre son nom et son mot de passe, le premier s'affiche, mais le second est masqué sous des astérisques (grâce à la propriété `PasswordChar` de la boîte de dialogue), pour protéger la saisie d'un œil fouineur. Votre programme a cependant accès au mot de passe tel qu'il a été saisi. Le mot de passe d'origine est *password* ; vous pouvez l'utiliser pour les tests. Le Listing 15.1 montre le code du module sous-jacent.

Listing 15.1 : La boîte de dialogue Connexion permet à l'utilisateur de se connecter à votre application

```
1: Option Explicit
2: Public LoginSucceeded As Boolean
3:
```

Listing 15.1 : La boîte de dialogue Connexion permet à l'utilisateur de se connecter à votre application *(suite)*

```
 4: Private Sub cmdCancel_Click()
 5:     ' Affecte la valeur False à la variable globale
 6:     ' pour indiquer l'échec de la connexion.
 7:     LoginSucceeded = False
 8:     Me.Hide
 9: End Sub
10:
11: Private Sub cmdOK_Click()
12:     ' Vérifie si le mot de passe est correct.
13:     If txtPassword = "password" Then
14:         ' Placer le code ici pour signaler
15:         ' à la procédure appelante la réussite de la fonction.
16:         ' Définir une variable globale est plus facile.
17:         LoginSucceeded = True
18:         Me.Hide
19:     Else
20:         MsgBox "Mot de passe non valide, réessayez !", , "Connexion"
21:         txtPassword.SetFocus
22:         SendKeys "{Home}+{End}"
23:     End If
24: End Sub
```

Le module de feuille utilise une variable globale nommée `LoginSucceeded` (déclarée à la ligne 2) dont la valeur `True` ou `False` peut être testée dans votre application au retour de la boîte de dialogue. Si l'utilisateur clique sur le bouton Annuler, la procédure `cmdCancel_Click()` paramètre `LoginSucceeded` à `False` à la ligne 7 et masque la feuille de connexion.

Pour adapter le code à vos besoins propres, suivez ces étapes :

1. Modifiez la chaîne de caractères du mot de passe dans la procédure événementielle `cmdOK_Click()` pour l'adapter à votre application. Le mot de passe est souvent placé dans un fichier et chiffré. Si vous enregistrez le fichier dans un fichier indexé, binaire ou de base de données, personne ne pourra lire le mot de passe à partir d'un simple éditeur de texte, comme c'est le cas avec un fichier texte.
2. Modifiez le texte de la boîte de message que vous voulez afficher si l'utilisateur se trompe de mot de passe.
3. Pour des raisons de sécurité, pensez à mettre la routine de contrôle du mot de passe dans une boucle `For` pour ne laisser à l'utilisateur qu'un nombre limité de tentatives et interdire ensuite l'affichage de la boîte de connexion. Cela rend plus difficile de percer le mot de passe.

Ce n'est pas parce que Microsoft a inséré une variable globale dans le code de la boîte de dialogue que cela en justifie l'emploi. Comme l'explique la remarque de la procédure `cmdOK_Click()`, la variable globale est la méthode la plus facile pour informer l'application du succès de la connexion, mais une programmation de qualité demande que vous remplaciez la variable globale par des variables locales. La meilleure manière de modifier ce code afin d'en améliorer la maintenance consiste sans doute à changer la sous-routine en une fonction booléenne. L'application appelante peut alors tester si la valeur de retour de la fonction est `True` ou `False`.

Le code à la fin de la procédure `cmdOK_Click()` peut paraître troublant, car son style diffère de ce à quoi vous avez été habitué — et vous y trouverez de nouvelles instructions. Jusqu'ici, `MsgBox ()` a été utilisé comme une fonction, mais le code contient l'instruction suivante :

```
MsgBox "Mot de passe non valide, réessayez !", , "Connexion"
```

Si Visual Basic supporte encore ce format de l'instruction `MsgBox`, Microsoft tente d'amener les programmeurs à utiliser de préférence la fonction `MsgBox()`. Pour la transformer en fonction, vous devez l'assigner à une variable (un type Variant conviendra) et ajouter des parenthèses, comme ceci :

```
varKeys = MsgBox("Mot de passe non valide, réessayez !", ,
➥"Connexion")
```

L'instruction `MsgBox` ne permet pas de déterminer le bouton de commande utilisé pour fermer la boîte de message. Par contre, la fonction `MsgBox()` renvoie le bouton sur lequel l'utilisateur a cliqué. Si OK est l'unique bouton affiché, il est inutile de tester la valeur retournée.

L'instruction suivante renvoie l'activité clavier à la zone de texte du mot de passe (cela ne survient que si l'utilisateur a saisi un mot de passe incorrect) par la méthode `SetFocus`. Cette méthode oblige l'application à rendre l'activité du clavier au contrôle auquel elle s'applique, même si elle doit normalement passer à un autre contrôle.

La dernière instruction utilise l'instruction `SendKeys` pour sélectionner le texte du mot de passe saisi par l'utilisateur, quelle qu'en soit la longueur. `SendKeys` déplace le curseur au début de la zone de texte, puis à la fin — ce qui a pour effet de sélectionner toute la zone. La première frappe de l'utilisateur remplacera le contenu précédent.

L'instruction `SendKeys` est expliquée en détail dans la leçon du Chapitre 7.

Boîte de dialogue Options

De tous les modèles de feuilles, la boîte de dialogue Options est celle qui effectue le moins de tâches par elle-même, mais qui a le plus de possibilités d'utilisations. Lorsque vous l'ajoutez, vous verrez le modèle illustré à la Figure 15.11. La boîte de dialogue comporte quatre pages, avec des onglets et un cadre dans le corps de chaque page. Vous pouvez ajouter des pages et des contrôles dans les cadres des pages, avec les options dont vous avez besoin.

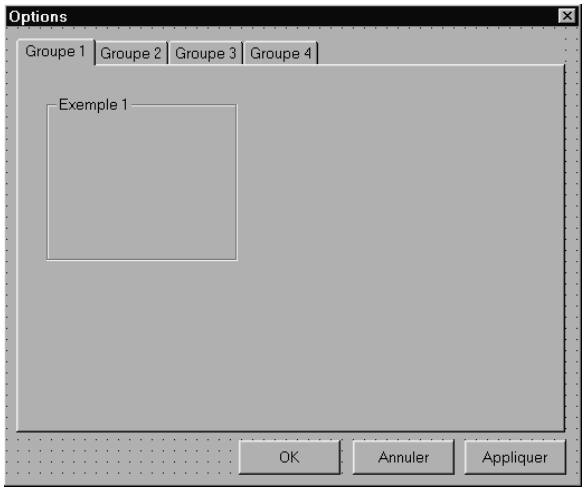


Figure 15.11
La boîte de dialogue Options affiche des pages pour présenter diverses options.

De nombreux programmes Windows contiennent une boîte de dialogue Options, à laquelle on accède par le menu Outils, qui ressemble beaucoup à celle créée par ce modèle. Ce n'est peut-être qu'un noyau, mais c'est le point de départ qui vous permettra de créer une boîte de dialogue plus complète.

La boîte de dialogue Options utilise un contrôle ActiveX particulier, la Barre d'onglets (*TabStrip*). Pour ajouter un contrôle Barre d'onglets à une application — sans passer par ce modèle de feuille — vous devez l'ajouter dans la boîte à outils à partir de la boîte de dialogue Composants. Dans le menu Projet, Composants, choisissez Microsoft Windows Common Controls 6.0.

Pour utiliser la boîte de dialogue Options, suivez ces règles générales :

- Ajoutez autant de pages d'options que vous en avez besoin. La manière la plus simple de modifier les onglets et les pages consiste à cliquer sur un des onglets puis sur les points de suspension de la propriété `(Personnalisé)`. La boîte de dialogue Pages de propriétés qui s'affiche vous aide à configurer les pages, les onglets et les

info-bulles que vous voulez utiliser dans la boîte de dialogue Options (voyez la Figure 15.12).

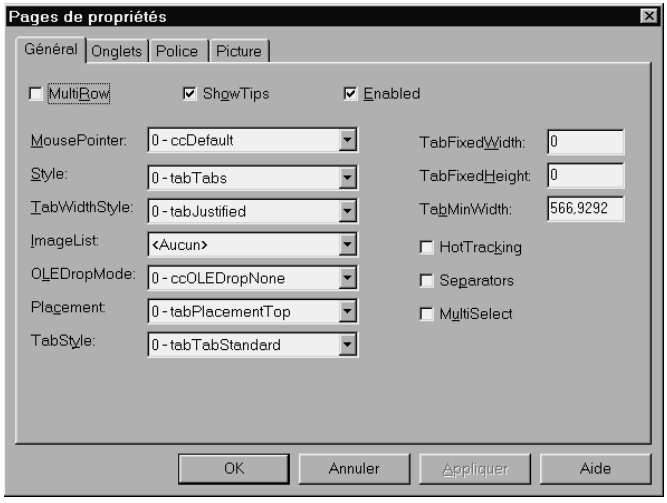


Figure 15.12
Utilisez la boîte de dialogue Pages de propriétés pour configurer les pages de la boîte de dialogue.

- Ajoutez une procédure générale qui lit tous les contrôles de la boîte de dialogue Options et configure les options correspondantes.
- Appelez la procédure de configuration des options à partir de la procédure `cmdApply_Click()` pour que les options prennent effet lorsque l'utilisateur clique sur le bouton Appliquer. (Vous pouvez aussi supprimer le bouton Appliquer et sa procédure associée si vous ne voulez pas que l'utilisateur ait accès à cette caractéristique.)
- Remplacez l'instruction suivante, dans la procédure événementielle `cmdOK _Click()`, par un appel à votre propre procédure de configuration des options :

```
MsgBox "Placez le code ici pour définir les options et fermer la
➥boîte de dialogue!"
```

- Modifiez la procédure événementielle `Form_KeyDown()` pour qu'elle gère l'ordre d'affichage des pages de la boîte de dialogue quand l'utilisateur se sert des touches Ctrl-Tab. Ce code n'est pas évident, car vous devez déterminer précisément le passage de l'activité d'un contrôle à l'autre par des instructions du programme.

La procédure événementielle `tbsOptions_Click()` affiche la bonne page (et masque les autres) dans le contrôle Barre d'onglets à l'exécution du contrôle.

Vous pouvez aussi mettre en pratique l'ajout de modèles de feuilles en utilisant l'assistant Création d'applications. Pour cela, créez un nouveau projet et démarrez l'assistant. Acceptez les valeurs par défaut jusqu'à ce que vous arriviez à la boîte de dialogue Feuilles standard. Cochez les quatre cases de modèles standards, puis double-cliquez sur Suivant, et enfin sur Terminer. Visual Basic crée le noyau de l'application. Une fois l'assistant achevé, cliquez sur OK pour lire les instructions de configuration avant de fermer la boîte de dialogue.

L'assistant Création d'applications ne crée qu'un noyau d'application. Vous devez compléter les détails. Cependant, ce noyau comporte les quatre modèles de feuilles standards proposés par défaut.

Testez l'application en l'exécutant pour voir ce qui fonctionne déjà. N'entrez pas de mot de passe (il est vierge par défaut tant que vous ne l'ajoutez pas au module de code), mais vous pouvez observer que l'écran d'accueil a récupéré le nom de l'utilisateur dans l'objet `App` et l'a automatiquement affiché dans la zone de texte Nom d'utilisateur.

L'écran d'accueil ne s'affiche que brièvement avant de laisser la place à la feuille principale. La boîte de dialogue A propos de s'affiche si vous choisissez A propos de dans le menu Aide, et la boîte de dialogue Options apparaît quand vous choisissez Options dans le menu Affichage. Ce projet, même si ce n'est qu'un noyau d'application, vous offre une quantité de code de qualité à étudier pour la mise en place de vos propres applications qui demandent un ou plusieurs modèles de feuilles standards.

L'assistant Création d'applications listera vos modèles personnels (si vous en avez créé) parmi les modèles standards fournis lorsque vous cliquez sur le bouton Modèles de feuilles dans la boîte de dialogue Feuilles standard.

Ecran Astuce du jour

Avez-vous déjà démarré un logiciel et été salué par une astuce sur la manière de mieux l'utiliser ? Windows 95 offre exactement ce type d'astuce (tant que vous ne coupez pas l'option d'affichage). Chaque fois que vous démarrez Windows 95, vous voyez une astuce différente. Pour interrompre cet affichage, décochez la case Afficher les astuces au démarrage. La Figure 15.13 montre le modèle de feuille Astuce du jour.

Lorsque vous ajoutez la boîte de dialogue Astuce du jour dans une fenêtre feuille, Visual Basic l'ajoute à la collection `Forms`. Selon la taille de votre écran et les paramètres de police par défaut, vous pouvez avoir à agrandir l'étiquette qui contient le texte Afficher les astuces au démarrage. Cliquez en dessous de l'étiquette Savez-vous… pour sélectionner l'étiquette `lblTipText`. C'est dans cette zone de texte que sera affichée l'astuce.

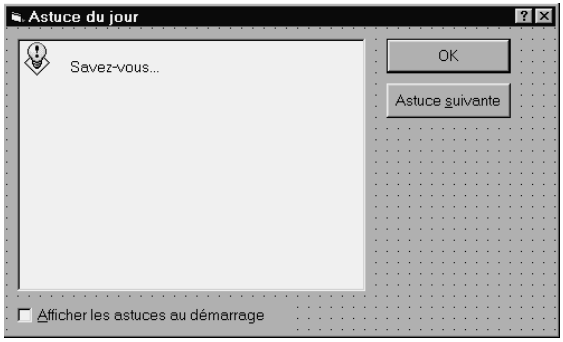


Figure 15.13
Une boîte de dialogue Astuce du jour peut fournir une assistance aux nouveaux utilisateurs du programme.

Le code du module configure la feuille pour afficher une nouvelle astuce du jour chaque fois que l'utilisateur démarre l'application. Les règles suivantes aident à comprendre les exigences liées à la boîte de dialogue Astuce du jour :

- Le code de la boîte de dialogue crée une nouvelle collection nommée `Tips`. La procédure y récupère l'astuce à afficher.
- Naturellement, la collection `Tips` doit lire les astuces dans un fichier que vous devez créer et fournir avec le projet. Il doit contenir une astuce par ligne.
- Le code charge le nom du fichier de la constante nommée `TIP_FILE` dans la fonction `Load_Tips()`. La procédure utilise la méthode `Add` pour ajouter chaque astuce à la collection en lisant le fichier. Votre seule véritable tâche consiste à créer ce fichier d'astuces à l'aide d'un éditeur de texte tel que le Bloc-notes de Windows.
- La procédure `DoNextTip()` sélectionne de manière aléatoire une astuce dans la collection et l'affiche à l'aide d'une méthode particulière, `DisplayCurrentTip` (qui est en fait une sous-routine se trouvant à la fin du code).
- La partie la plus technique du code est la plus courte. Elle se trouve dans la procédure `chkLoadTipsAtStartup()`. Microsoft en fournit heureusement le code. Il utilise la commande `SaveSetting` pour modifier la base de registres du système. Elle enregistre dans ce cas la valeur qui détermine s'il faut ou non afficher cette feuille au démarrage. Si la case Afficher les astuces au démarrage est décochée, la base de registres est modifiée en conséquence et le code n'affichera plus les astuces dans les sessions suivantes.

Le modèle de feuille Astuce du jour est sans doute le plus courant des modèles qui restent à décrire dans cette leçon. Suivez ces étapes pour mettre en pratique la configuration d'une boîte de dialogue Astuce du jour sur votre système :

1. Démarrez le Bloc-notes de Windows. Créez un fichier `Tipofday.txt` (c'est le nom par défaut utilisé par la boîte de dialogue Astuce du jour). Enregistrez ce fichier dans le répertoire de l'application. Saisissez ensuite le contenu suivant, puis enregistrez le fichier et quittez le Bloc-notes :

```
Brosser vos dents tous les jours et vos chaussures toutes les semaines.
Epargner pour votre retraite (personne ne s'en soucie plus que vous).
Dormir beaucoup est un excellent traitement de l'insomnie.
Lire un bon livre avant de voir le film.
Ne pas conduire en casse-cou… ou casse-briques ; conduisez sans casse.
Faire de l'exercice plus souvent que vous prenez un dessert.
```

2. Créez une nouvelle application et affichez la fenêtre de feuille standard. Avant tout, enregistrez la feuille et le projet dans le même répertoire que le fichier des astuces. C'est nécessaire pour permettre au code de les trouver.
3. Dans le menu Projet, choisissez Ajouter une feuille, puis Astuce du jour.
4. Modifiez la propriété `Caption` `Savez-vous…` de l'étiquette en `N'oubliez pas de….`
5. Exécutez l'application pour voir les résultats.

Pas de boîte de dialogue Astuce du jour ? Vous avez bien créé le fichier des astuces et ajouté la boîte de dialogue à votre application, mais Visual Basic affiche d'abord la feuille principale du projet (`Form1`, si vous n'avez pas modifié son nom). Il reste à définir la feuille Astuce du jour comme feuille de démarrage et configurer le code qui affiche la feuille normale quand l'utilisateur ferme la boîte de dialogue Astuce du jour.

Dans le menu Projet, sélectionnez Propriétés, et configurez `frmTip` comme Objet de démarrage. Cliquez sur OK pour fermer la boîte de dialogue. L'application est désormais configurée pour afficher la boîte de dialogue Astuce du jour au démarrage.

Vous devez connecter la feuille principale (`Form1`) à la boîte de dialogue Astuce du jour pour qu'elle s'affiche une fois cette dernière fermée. Modifiez la procédure `cmdOK_Click()` comme suit :

```
1: Private Sub cmdOK_Click()
2:     Unload Me    ' Ferme la boîte de dialogue Astuce du jour
3:     Form1.Show   ' Montre la feuille principale
4: End Sub
```

Il reste une chose à faire. Si l'utilisateur décide de ne plus afficher les astuces au démarrage, il n'y a alors plus moyen d'afficher la feuille principale. Vous devez donc ajouter à la procédure `Form_Load()` une instruction `Form1.show`, comme ceci :

```
1: Private Sub Form_Load()
2:     Dim ShowAtStartup As Long
3:
```

```
 4:     ' Voit si la feuille doit apparaître au démarrage.
 5:     ShowAtStartup = GetSetting(App.EXEName, "Options", "Show
        ➥Tips at Startup", 1)
 6:     If ShowAtStartup = 0 Then
 7:         Unload Me
 8:         Form1.Show  'Montre la feuille normale ** Nouvelle instruction
 9:         Exit Sub
10:     End If' La suite du code n'est pas reprise ici
```

Vous pouvez maintenant exécuter l'application et lire les astuces de manière aléatoire en cliquant sur le bouton de commande Astuce suivante. Quand vous cliquez sur OK, la feuille principale s'affiche. Si un utilisateur décide de ne plus consulter les astuces dans les sessions suivantes, la feuille principale s'affiche directement au démarrage.

Vous pouvez aussi configurer l'application pour qu'elle affiche l'astuce et la feuille normale d'autres façons. Par exemple, en ajoutant les méthodes `Show` *adéquates à une sous-routine* `Main` *et en utilisant cette sous-routine comme objet de démarrage.*

Boîte de dialogue Connexion ODBC

ODBC fournit un ensemble standard de commandes permettant l'accès à différents types de données enregistrées sur des ordinateurs de types variés. Le but du standard ODBC est de permettre à de nombreux types de systèmes d'accéder à des données stockées ailleurs.

La Figure 15.14 montre le modèle de feuille qui s'affiche lorsque vous insérez la boîte de connexion ODBC dans votre fenêtre feuille.

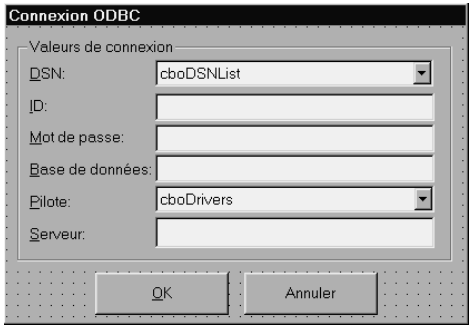


Figure 15.14
Le modèle de feuille Connexion ODBC vous permet de configurer un accès à une base de données externe.

La boîte de dialogue Connexion ODBC peut être ajoutée à une application pour permettre aux utilisateurs de sélectionner une source de données ODBC et d'y accéder. La source indique l'emplacement et le type des données auxquelles l'application veut accéder. Le Tableau 15.1 décrit chaque champ ODBC.

Tableau 15.1 : Les zones de texte de la boîte de dialogue Connexion ODBC spécifient les options ODBC

Nom	*Description*
DSN	Nom de la source de données. Cette option liste (dans une boîte de liste déroulante) les sources ODBC actuellement enregistrées dans la base de registres de l'utilisateur.
ID	Identifiant de l'utilisateur tel qu'il est défini dans la base de données ODBC, pour lui permettre de valider la connexion.
Mot de passe	Mot de passe de l'utilisateur permettant d'accéder au système.
Base de données	Nom de la base de données à laquelle il faut se connecter.
Pilote	Liste déroulante qui présente tous les pilotes présents sur le système et qui permet à l'utilisateur d'enregistrer un nouveau pilote ODBC.
Serveur	Nom du serveur qui fournit la base de données si la DSN n'est pas disponible.

Le code nécessaire pour connecter les valeurs contenues dans la boîte de dialogue ODBC à la bonne base de données compatible ODBC est assez complet. Vous devez comprendre le sens des commandes ODBC nécessaires à la connexion de la base de données externe. Elles dépassent cependant le cadre de cet ouvrage, car les programmeurs Visual Basic ont rarement à recourir à de telles routines, sauf dans les applications système.

Ajouter vos propres modèles de feuilles

Il est facile d'ajouter vos propres feuilles à la collection de Visual Basic. Une fois que vous avez créé une feuille que vous souhaitez ajouter aux modèles, enregistrez-la dans le dossier `\Template\Forms`. Chaque fois que vous ouvrirez la liste des modèles, même à partir de l'assistant Création d'applications, la feuille apparaîtra désormais avec les autres.

N'ajoutez au dossier des modèles que des feuilles assez générales, sans trop de détails, à moins que vous ne souhaitiez expressément voir tous ces détails dans les feuilles que vous créerez à partir du modèle.

Pour supprimer une feuille ajoutée dans le dossier des modèles, démarrez l'Explorateur Windows et allez dans le dossier `\Template\Forms`. *Sélectionnez la feuille et supprimez-la. La prochaine fois que vous afficherez la liste des modèles, elle aura disparu.*

En résumé

La leçon d'aujourd'hui vous a expliqué comment utiliser les modèles de feuilles pour normaliser les applications et accélérer le développement et la précision des programmes. Les modèles de feuilles comportent plusieurs feuilles standards que les programmeurs ajoutent souvent aux applications Windows. Vos utilisateurs apprécieront de voir des feuilles normalisées dans vos applications.

Les modèles sont des feuilles d'usage général qui peuvent s'ajouter à tout projet. Ils contiennent des réceptacles pour le texte et les images qui peuvent êtres modifiés une fois que le modèle est chargé dans le projet. Votre application commande l'affichage de la feuille et interagit avec le code du modèle de feuille.

La leçon de demain débute une étude sur deux jours des objets et de leur relation à la programmation Visual Basic.

Questions-réponses

Q Pourquoi dois-je rendre les modèles de feuilles généraux ?

R En rendant vos modèles de feuilles aussi généraux que possibles (en ne conservant que les détails qui ne changent pas d'une application à l'autre), vous simplifiez la génération des nouvelles feuilles à partir des modèles. Dans la leçon d'aujourd'hui, vous avez vu plusieurs modèles fournis avec Visual Basic. La plupart des textes qu'ils contiennent sont des réceptacles qui indiquent où il faut que vous interveniez pour personnaliser la feuille.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités et des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se trouvent à l'Annexe A.

Quiz

1. Quel est le but des modèles de feuilles ?
2. Décrivez deux manières d'ajouter des modèles de feuilles à vos applications.
3. Décrivez le code nécessaire pour connecter la boîte de dialogue A propos de à un projet.
4. Vrai ou faux. Vous devez écrire le code qui affiche les informations système quand l'utilisateur clique sur le bouton Infos système de la boîte de dialogue A propos de.
5. Quelle est la différence entre un écran d'accueil et une feuille normale ?
6. Considéreriez-vous que la boîte de dialogue Astuce du jour est un écran d'accueil ?
7. Quel est le but de la commande `SaveSetting` ?
8. Quelle est la signification d'ODBC et quel est son but ?
9. Que devez-vous faire dans la boîte de dialogue Propriétés pour que votre application puisse afficher correctement un écran d'accueil ou une boîte de dialogue Astuce du jour ?
10. Décrivez le format du fichier des astuces nécessaire à la boîte de dialogue Astuce du jour.

Exercices

1. Suivez le conseil donné dans la partie concernant la boîte de dialogue Connexion pour en transformer le code en un meilleur ensemble de routines. Remplacez la variable globale par des variables locales.
2. Créez une application qui affiche sur le PC une astuce différente chaque fois que l'application démarre. (Vous pouvez modifier le fichier des astuces décrit dans la leçon d'aujourd'hui.) Ajoutez une option de menu à la feuille principale pour que les astuces puissent se réafficher au démarrage. *Indice* : regardez la procédure `chkLoadTipsAtStartup()` et utilisez la commande `SaveSetting` pour réinitialiser les astuces. Même si vous ne maîtrisez pas cette commande, vous disposez de tous les outils nécessaires à l'achèvement rapide de ce projet.

Chapitre 16

Visual Basic et les objets

Ce chapitre vous montre comment travailler avec les objets dans Visual Basic. Vous avez déjà utilisé certains objets en manipulant les feuilles, les contrôles et l'objet `Printer`. Nous étendrons aujourd'hui cette connaissance ; vous apprendrez un nouveau contrôle nommé contrôle OLE, qui permet d'utiliser des objets externes à l'environnement Visual Basic.

En outre, vous étudierez plus en profondeur les objets prédéfinis de Visual Basic, tels que `Screen` ou `App`. Ils envoient à vos applications des informations que vous pouvez utiliser pour prendre des décisions ou afficher des titres et des informations utilisateur sur la feuille. Une fois que vous saurez comment utiliser les objets prédéfinis, vous apprendrez comment travailler avec les collections d'objets.

Vous apprendrez aujourd'hui :

- le contrôle OLE ;
- les différences entre la liaison et l'incorporation d'objet ;
- comment placer les objets d'autres applications dans vos applications ;
- l'activation in situ ;
- comment utiliser les groupes de contrôles ;
- comment gérer des collections, ;
- comment gérer l'Explorateur d'objets.

OLE pour les objets externes

OLE est un terme familier aux utilisateurs et aux programmeurs de Windows. Vous pouvez incorporer des objets OLE dans vos applications pour augmenter la puissance de vos programmes et réduire la programmation. En utilisant des objets OLE déjà définis par d'autres applications, vous tirez avantage de la réutilisation d'objets.

OLE (Object Linking and Embedding) signifie Liaison et incorporation d'objets. De nombreuses applications Windows offrent leurs données sous forme d'objets OLE, que vous pouvez incorporer dans d'autres applications compatibles OLE.

La technologie ActiveX remplace rapidement OLE. En fait, Microsoft appelle les contrôles ActiveX les "contrôles anciennement OLE". En tout cas, OLE est toujours là et de nombreuses applications Visual Basic l'utilisent encore, comme le prouve la décision de Microsoft de conserver par défaut le contrôle OLE dans la fenêtre Boîte à outils. La leçon d'aujourd'hui est non seulement importante en soi, mais elle constitue aussi une excellente introduction à la leçon de demain qui concerne les objets ActiveX.

La raison d'être de l'utilisation d'OLE réside dans la possibilité d'employer des objets extérieurs dans les applications que vous concevez. Le contrôle OLE de la fenêtre Boîte à outils gère la connexion à l'objet OLE de votre projet. Les utilisateurs peuvent tirer avantage de l'activation in situ lorsqu'ils accèdent à des objets OLE incorporés.

L'activation in situ fait référence à la capacité de modifier un objet à l'intérieur d'une application Visual Basic en utilisant les menus et les commandes de l'application parente. Quand l'objet OLE est incorporé dans une application, vous n'avez pas à écrire d'options de menu pour guider l'utilisateur dans la modification de l'objet — ceux de l'application d'origine apparaîtront automatiquement.

Liaison et incorporation

Le contrôle OLE peut, suivant sa configuration, maintenir un lien vers un objet ou l'incorporer. Lorsque vous établissez un lien vers une autre application, le contrôle OLE (le *conteneur*) contient une liaison avec un de ses documents. Si ce dernier est modifié dans l'application d'origine, votre application le reflétera. L'objet est dit *permanent*, car son contenu peut être maintenu à jour par la liaison.

Un contrôle conteneur est un contrôle OLE qui contient l'objet de données d'une autre application.

Un objet permanent est un objet extérieur à votre projet qui ne disparaît pas simplement parce que votre application se termine.

Lorsque vous incorporez dans votre application un objet OLE, le contrôle conteneur OLE contient une copie de l'objet document créé dans l'autre application compatible OLE. Mais aucune liaison n'est maintenue entre votre application et l'objet d'origine. Donc, si ce dernier est modifié dans son application originelle, cela ne se reflétera pas dans votre application, car vous disposez d'une copie de cet objet.

Voici comment les deux types d'activité OLE affectent vos projets :

- Si vous liez un objet au contrôle OLE de votre application, Visual Basic démarre en fait l'application parente lorsque l'utilisateur tente d'utiliser ou de modifier l'objet en double-cliquant dessus. L'utilisateur doit choisir l'option de menu Fichier, Fermer et revenir à l'application lorsqu'il a fini de travailler sur l'objet.
- Si vous incorporez un objet dans le contrôle OLE de votre application, il en fait partie intégrante, car il réside dans votre application.

Le contrôle OLE

Ouvrez une nouvelle application et double-cliquez sur le contrôle OLE pour l'ajouter à la feuille. Une grande zone blanche s'affiche un instant sur la feuille, puis la boîte de dialogue Insérer un objet, illustrée à la Figure 16.1, s'affiche. Certains contrôles affichent une boîte de dialogue lorsque vous cliquez sur la rubrique de propriété `(Personnalisé)`, mais le contrôle OLE est le seul qui affiche automatiquement une boîte de dialogue dès qu'il est placé sur la feuille.

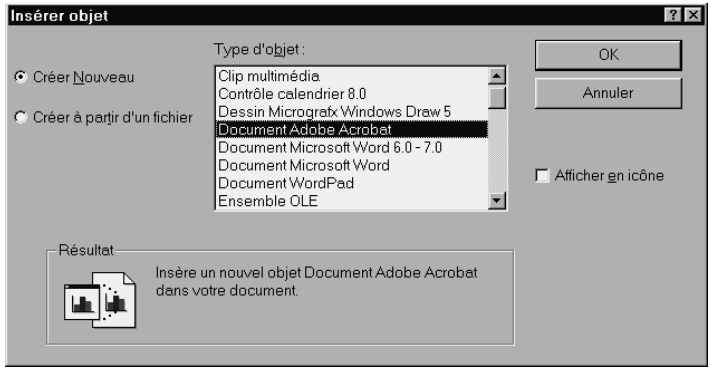


Figure 16.1
La boîte de dialogue Insérer un objet contient toutes les applications OLE inscrites sur votre système.

La zone de liste Type d'objet contient les applications compatibles OLE parmi lesquelles vous devez faire votre sélection. La liste varie suivant les logiciels installés sur votre machine. Une application met à jour la base de registres avec ses objets lors de son installation. Une application Windows peut produire deux ou plusieurs contrôles personnalisés. Par exemple, si PowerPoint 97 est installé, vous trouverez deux contrôles PowerPoint dans la liste.

L'option Créer nouveau vous permet de spécifier une application pouvant créer le type d'objet que vous voulez incorporer. Si l'objet de données de l'autre application existe déjà, vous pouvez cliquer sur l'option Créer à partir d'un fichier. Une page de sélection de fichier s'affiche alors (voyez la Figure 16.2). A partir de cette fenêtre, vous pouvez incorporer l'objet ou cocher la case Lien pour ajouter un pointeur sur l'objet.

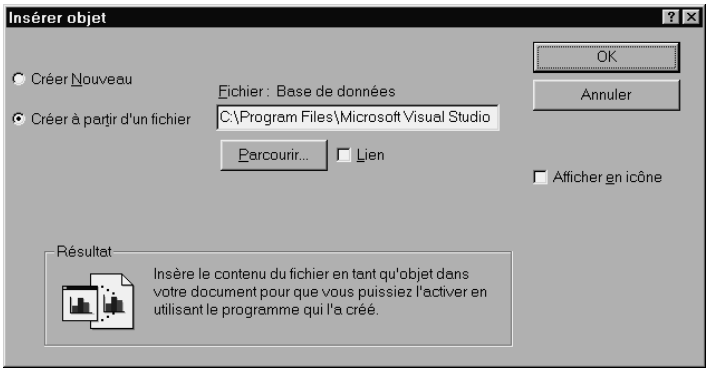


Figure 16.2
Vous pouvez incorporer ou lier un objet existant dans votre contrôle OLE.

Supposons que vous voulez créer des notes dans une application, qui peuvent par exemple se rapporter à un enregistrement client. Vous pouvez :

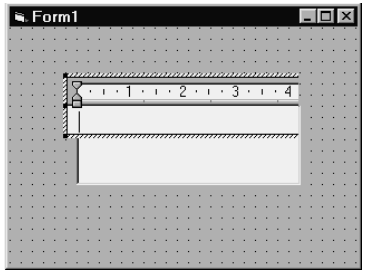
- laisser l'utilisateur saisir la note dans un simple contrôle zone de texte ;
- améliorer le concept de zone de texte simple en écrivant une routine de capture des frappes clavier qui analyse la saisie et remplace la zone de texte simple en un traitement de texte complet qui formate le texte et supporte les règles et les marges (ce qui peut être assez long à écrire) ;
- Incorporer un objet document WordPad à l'aide d'OLE.

A l'évidence, la troisième option paraît la bonne, car elle vous permet d'offrir à l'utilisateur la puissance d'un traitement de texte sans que vous ayez à en écrire une ligne : WordPad existe déjà. Tout le monde ne dispose pas de Microsoft Word sur sa machine, mais WordPad est fourni avec Windows (depuis Windows 95). Vos utilisateurs y ont donc accès, ce qui leur permet d'effectuer l'activation in situ de l'objet.

Pour incorporer un document WordPad, suivez ces étapes :

1. Après avoir inséré le contrôle OLE sur la feuille, sélectionnez l'option Créer nouveau.
2. Faites défiler la liste jusqu'à ce que vous trouviez la rubrique Document WordPad.
3. Double-cliquez sur cette entrée pour incorporer un objet Document WordPad dans votre application. Vous verrez une partie de la règle de WordPad sur le contrôle OLE, comme l'illustre la Figure 16.3.

Figure 16.3
Le contrôle OLE contient maintenant un objet WordPad.



4. Modifiez la propriété `SizeMode` du contrôle OLE en `1 - Stretch` pour que l'objet WordPad s'adapte à la taille du contrôle OLE.
5. Augmentez les propriétés `Width` et `Height` de la feuille à, respectivement, `6945` et `5670`.
6. Configurez les propriétés suivantes du contrôle OLE :

```
Height : 3375
Left : 840
Top : 1080
Width : 5055
```

   Vous pourriez ajouter d'autres éléments à ce projet et donner un autre nom à la feuille, mais ne vous en préoccupez pas pour l'instant.
7. Exécutez l'application. Au démarrage, rien de particulier ne semble se produire. Le contour du contrôle OLE est affiché au centre de la feuille.
8. Double-cliquez sur le contrôle OLE. Comme le montre la Figure 16.4, un menu s'affiche, lié à l'activation in situ du document WordPad incorporé, et une règle.

Il existe plusieurs autres manières d'activer un objet OLE, par exemple dès qu'il obtient l'activité du clavier. La propriété `AutoActivate` commande la manière d'activer l'objet. Sa valeur par défaut est `2 - DoubleClick`.

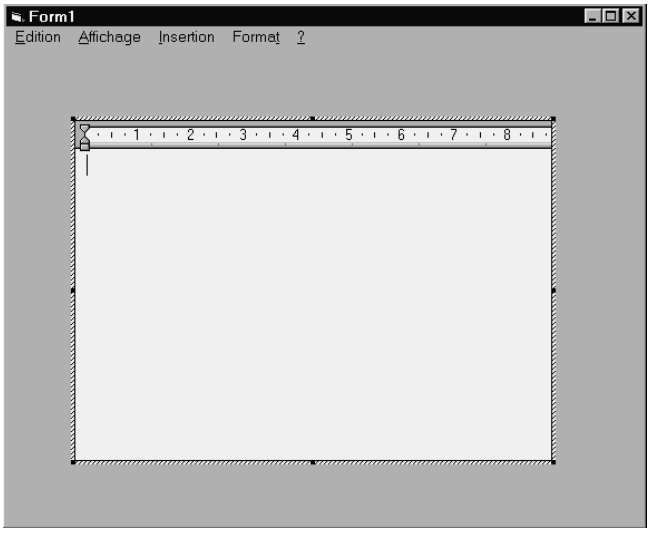


Figure 16.4
Voici un traitement de texte au milieu de votre application Visual Basic !

9. Tapez du texte et utilisez les menus pour le formater à votre convenance. Toutes les options de WordPad sont disponibles dans votre application Visual Basic.

10. Fermez l'application.

Aucun enregistrement de ce que vous avez tapé dans WordPad n'est effectué. Lorsque vous fermez l'application Visual Basic, le document WordPad s'évanouit et n'est jamais enregistré. Ni Visual Basic, ni WordPad ne vous en avertissent. Cet enregistrement vous incombe en tant que programmeur. Notez que les menus in situ ne comportent pas les options habituelles du menu Fichier.

Enregistrer le contenu de l'objet

Microsoft conseille d'enregistrer et de charger les objets incorporés à l'aide d'une des méthodes d'enregistrement de fichier. Votre application ayant la charge de contenir l'objet incorporé, l'application source n'a donc aucune autorité pour l'enregistrer dans votre application.

Utilisez la méthode `SaveToFile` pour enregistrer les données de votre contrôle conteneur OLE. Le Listing 16.1 montre un exemple de code qui enregistre les modifications dans un fichier binaire particulier. Un fichier binaire est plus restrictif que les types de fichiers séquentiels ou indexés que vous avez déjà abordés, mais il offre une méthode rapide et efficace pour enregistrer des objets, tant que vous les relisez ensuite dans le même ordre.

Listing 16.1 : Enregistrer l'objet contrôle conteneur OLE sur disque

```
 1:  Dim intFileNum as Integer
 2:
 3:  ' Obtenir le premier numéro de fichier disponible
 4:  intFileNum = FreeFile
 5:
 6:  ' Ouvrir le fichier de sortie
 7:  Open "TEST.OLE" For Binary As #intFileNum
 8:
 9:  ' Enregistrer le fichier
10: oleObj1.SaveToFile intFileNum
11:
12: ' Fermer le fichier
13: Close
```

Lors des exécutions suivantes, votre application doit lire les dernières valeurs de l'objet dans le contrôle conteneur OLE en utilisant la méthode `ReadFromFile`, illustrée dans le Listing 16.2.

Listing 16.2 : Lire le contenu de l'objet contrôle conteneur OLE enregistré à la précédente exécution

```
 1:  Dim intFileNum as Integer
 2:
 3:  ' Obtenir le premier numéro de fichier disponible
 4:  intFileNum = FreeFile
 5:
 6:  ' Ouvrir le fichier de sortie
 7:  Open "TEST.OLE" For Binary As #intFileNum
 8:
 9:  ' Lire le fichier dans l'objet
10: oleObj1.ReadFromFile intFileNum
11:
12: ' Fermer le fichier
13: Close
```

Vous pouvez placer le code de lecture et d'écriture dans des procédures événementielles liées à des boutons de commande ou des éléments de menu. Répétons que c'est votre application, et non pas l'application OLE, qui a la charge d'enregistrer et de charger les données. L'application OLE s'occupe quant à elle de toutes les autres tâches en relation avec l'objet.

Travailler avec les objets

Peut-être avez-vous déjà entendu le terme *programmation orientée objet*, ou *POO*. Visual Basic n'est pas strictement un langage orienté objet, même s'il supporte les objets en de nombreuses façons. La leçon d'aujourd'hui vous aidera à mieux comprendre les objets de Visual Basic et leur relation aux groupes et aux collections.

La programmation orientée objet ou POO est une programmation à l'aide d'éléments de données qui représentent des objets ayant des méthodes et des propriétés proches de celles des objets de Visual Basic. La véritable POO contient des objets qui peuvent hériter d'autres objets. Visual Basic ne supporte pas complètement et exactement le modèle objet, même si, dans la pratique, il paraît en remplir les buts ultimes mieux qu'aucun langage purement POO ne l'a fait.

Les sections qui suivent décrivent des objets non OLE. Vous apprendrez comment affûter vos compétences en programmation en tirant avantage des groupes de contrôles et des collections d'objets.

Un groupe de contrôles est un tableau de contrôles auxquels on accède à l'aide d'indices.

Programmer avec des objets

Dans le Chapitre 8, vous avez appris à utiliser l'instruction `If TypeOf` pour tester la forme d'un objet. Nous expliquerons aujourd'hui plus en détail ce que fait cette instruction — elle détermine en réalité la *classe* d'un objet, mais nous y reviendrons.

Un *objet* peut être à peu près n'importe quoi dans Visual Basic. Vous avez déjà travaillé avec des objets tels que les contrôles, les feuilles et l'objet `Printer`. Vous avez aussi transmis le type de données `Object` à une procédure dans le Chapitre 13. Vous savez déjà que les *objets contrôles* contiennent des propriétés, des méthodes et des événements. Les contrôles contiennent du code et des éléments de données. D'une certaine façon, un objet — tel que le contrôle bouton de commande — est comme un paquet qui vous est tendu par les développeurs de Visual Basic. Vous n'avez pas à écrire le code qui déclenche un bouton de commande, à déclarer des variables pour décrire son apparence et son titre, etc. Les méthodes encapsulées du bouton de commande font tout le travail à votre place.

Les objets sont encapsulés. Comme une capsule contient des médicaments ou des astronautes en route vers la lune, des méthodes, des événements et des propriétés sont encapsulées dans un objet. L'encapsulation vous permet

de travailler sur les objets d'un point de vue plus élevé en vous évitant l'écriture de tout le code nécessaire à leur support. Une classe définit le comportement et l'apparence des objets qu'elle contient.

Classes d'objets

Les objets n'apportent pas seulement l'encapsulation, ils font aussi partie d'une hiérarchie nommée *classe* d'objets. L'avantage d'une classe est que tous ses objets partagent les mêmes caractéristiques. (Un objet unique est dit être une *instance* de la classe.)

Vous pouvez créer vos propres objets. Si vous les intégrez dans une classe existante, ils bénéficient automatiquement, ou héritent, des nombreux propriétés, méthodes et événements de la classe.

Lorsque vous testez un objet à l'aide de `If TypeOf`, Visual Basic renvoie sa classe. La ligne de code suivante renvoie donc `True` ou `False`, suivant que l'objet `myObj` fait partie ou pas de la classe `CommandButton` :

```
If TypeOf myObj Is CommandButton    'Vérifie la classe
```

Visual Basic supporte aussi la fonction `TypeOf()` qui renvoie le nom de la classe. Par exemple, `TypeOf(myObj)` pourrait renvoyer `CommandButton` ou `Form`.

Les classes rendent plus souple la programmation avec les objets. Par exemple, les instructions `With … End With` vous permettent d'assigner simplement plusieurs propriétés à un objet. Notez la redondance du code ci-dessous :

```
chkMaster.Caption = "Source principale"
chkMaster.Alignment = vbLeftJustify
chkMaster.Enabled = True
chkMaster.Font.Bold = False
chkMaster.Left = 1000
chkMaster.RightToLeft = False
chkMaster.Top = 400
```

En entourant un objet d'un bloc `With … End With`, vous pouvez éliminer la répétition du nom de l'objet. Le code suivant est identique au précédent :

```
With chkMaster
   .Caption = "Source principale"
   .Alignment = vbLeftJustify
   .Enabled = True
   .Font.Bold = False
```

```
    .Left = 1000
    .RightToLeft = False
    .Top = 400
End With
```

L'utilisation de `With … End With` sur deux ou trois propriétés demande plus de saisie que des assignations directes. Cependant, lorsque vous avez plus de trois propriétés à assigner, la clause `With` devient une instruction attirante, car elle demande moins de saisie et la maintenance est plus facile si vous devez ajouter des propriétés.

Si vous pensez avoir à assigner des propriétés supplémentaires dans de futures versions du programme, vous pouvez anticiper et utiliser la clause `With`, même avec une ou deux propriétés.

Objets système

Contrairement aux objets que vous déclarez, les objets système sont les objets comme `App` ou `Printer`, que vous avez déjà utilisés dans ce livre. Si vous ne pouvez pas transmettre les objets système (ils sont par nature globaux), vous pouvez les traiter comme les objets que vous créez. Les objets système représentent des éléments particuliers de votre application.

Le Tableau 16.1 décrit les objets système et liste quelques méthodes importantes qui peuvent leur être appliquées.

Tableau 16.1 : Les objets système supportent plusieurs méthodes

Objet système	*Méthodes*	*Description*
`App`		Application en cours.
	`EXEName`	Renvoie le nom de fichier de l'application.
	`Path`	Renvoie le chemin d'accès de l'application.
	`Title`	Renvoie le texte de la barre de titre de la fenêtre principale.
	`PrevInstance`	Renvoie `True` ou `False`, selon qu'une autre copie de l'application est en cours d'exécution ou pas.
`Clipboard`		Presse-papiers de Windows.
	`Clear`	Efface le Presse-papiers.
	`GetData`	Renvoie l'image enregistrée dans le Presse-papiers.

Tableau 16.1 : Les objets système supportent plusieurs méthodes *(suite)*

Objet système	*Méthodes*	*Description*
	`GetFormat`	Renvoie le format de l'objet dans le Presse-papiers.
	`GetText`	Renvoie le texte du Presse-papiers.
	`SetData`	Copie une image dans le Presse-papiers.
	`SetText`	Copie du texte dans le Presse-papiers.
	`SelStart`	Utilisé pour des opérations de sélection dans le Presse-papiers.
	`SelLength`	Utilisé pour des opérations de sélection dans le Presse-papiers.
	`SelText`	Utilisé pour des opérations de sélection dans le Presse-papiers.
`Debug`		Fenêtre Exécution.
	`Print`	Copie les informations à l'exécution dans la fenêtre Exécution (utilisable uniquement pour les programmes Visual Basic non-EXE exécutés depuis l'environnement de développement).
`Err`		Contient des informations sur l'état d'erreur actuel de l'application.
		La propriété essentielle `Number` contient un code d'erreur qui correspond à l'erreur système la plus récente (zéro si aucune erreur ne s'est produite).
`Printer`		Imprimante système. Ses méthodes ont été présentées dans le Chapitre 13.
`Screen`		Ecran de l'utilisateur.
	`FontCount`	Renvoie le nombre de polices supportées par l'écran actuel.
	`Fonts`	Contient une liste de tous les noms des polices possibles à l'écran.
	`Height`	Renvoie la hauteur de la zone écran en twips.

Tableau 16.1 : Les objets système supportent plusieurs méthodes *(suite)*

Objet système	*Méthodes*	*Description*
	`MousePointer`	Contient la forme du curseur de la souris et détermine sa forme si vous spécifiez un autre curseur.
	`TwipsPerPixelX`	Renvoie le nombre de twips possibles en horizontal.
	`TwipsPerPixelY`	Renvoie le nombre de twips possibles en vertical.
	`Width`	Renvoie la largeur de l'écran en twips.

Vous avez travaillé avec la majorité des objets système — en particulier `Printer` et `Screen` — avant la leçon d'aujourd'hui. L'objet `App` est utile pour déterminer au moment de l'exécution des informations comme le chemin et le nom de fichier du programme ; l'objet `Clipboard` fournit quelques fonctionnalités intéressantes à utiliser. L'objet `Debug` vous permet d'interagir avec votre programme lors des tests pour vous aider à en extraire les bogues.

Suivant les besoins de votre application, l'objet `Clipboard` est relativement simple à programmer. C'est le Presse-papiers de Windows ; vos applications peuvent donc y couper ou copier des informations, que l'utilisateur peut ensuite coller dans une autre application Windows. Votre application peut aussi coller des informations déjà contenues dans le Presse-papiers.

Vous pouvez utiliser l'objet `Clipboard` et ses propriétés pour sélectionner du texte à partir de votre programme et pour déterminer le texte sélectionné par les utilisateurs. Par exemple, la propriété `SelStart` marque la position du début de la sélection dans la zone de texte (ou d'un autre type de contrôle qui reçoit la sélection). La mise à `0` de `SelStart` place le curseur devant le premier caractère. `SelLength` détermine le nombre de caractères choisis. Si vous sélectionnez du texte en configurant `SelStart` et `SelLength`, ce texte est transmis au Presse-papiers lorsque l'utilisateur frappe Ctrl-C (pour copier) ou Ctrl-X (pour couper). `SelText` est une chaîne qui contient le texte sélectionné.

Si vous avez sélectionné du texte dans une zone de texte, il se retrouve dans `SelText`. Vous pouvez effacer la valeur courante de l'objet `Clipboard` (le Presse-papiers ne peut contenir qu'une valeur à la fois), puis envoyer le texte sélectionné au Presse-papiers à l'aide du code suivant :

```
Clipboard.Clear    ' Efface le contenu du Presse-papiers
Clipboard.SetText txtName.SelText  'Copie le texte
```

Pour copier le texte du Presse-papiers dans une variable, vous pouvez utiliser `GetText()`, ainsi :

```
strInfo = Clipboard.GetText()
```

Pour remplacer le texte sélectionné dans un contrôle par le texte contenu dans l'objet `Clipboard`, vous pouvez le faire de cette manière :

```
txtName.SelText = Clipboard.GetText()
```

La méthode `GetText()` utilise parfois des arguments et elle exige des parenthèses, même si vous n'en spécifiez pas. Pour travailler sur du texte dans le Presse-papiers, il n'est pas nécessaire de fournir des arguments.

Groupes d'objets et de contrôles

Une des choses les plus intéressantes à faire avec les objets est de déclarer un groupe d'objets, comme un groupe de boutons de commande ou de feuilles. En outre, ces objets n'ont même pas besoin d'exister. Il n'est par exemple pas obligatoire de déclarer toutes les feuilles au moment de la conception, car vous pouvez toujours créer un groupe de feuilles à l'exécution.

Vous connaissez déjà les collections `Forms` et `Printers`. Visual Basic supporte également la collection `Controls`, qui vous permet de parcourir tous les contrôles comme s'ils étaient des variables de tableaux. Par exemple, le code qui suit masque tous les contrôles :

```
For intCtr = 0 To Controls.Count - 1
  Controls(intCtr).Visible = False
Next intCtr
```

Si votre application contient plusieurs feuilles, vous pouvez masquer tous les contrôles de l'ensemble des feuilles en utilisant des boucles imbriquées (l'utilisation de `For Each` élimine la nécessité de recourir à `Count - 1`) :

```
1:  Dim frmAForm As Form
2:  Dim ctlAControl As Control
3:  For Each frmAForm In Forms   ' Parcours de toutes les
 ➥feuilles
4:    For Each ctlAControl In frmAForm.Controls
5:      ctlAControl.Visible = False
6:    Next ctlAControl
7:  Next frmAForm
```

Un menu est considéré comme un contrôle dans la collection `Controls`. *Dans de nombreux cas, il est souhaitable d'omettre les menus dans une boucle de ce type en testant, à l'aide de la fonction* `TypeOf()`, *si le contrôle est un objet* `Menu` *avant de le masquer.*

La collection `Controls` contient tous les contrôles de la feuille en cours ; vous pouvez cependant définir un groupe de contrôles d'un type particulier. Un groupe de contrôles se déclare comme suit :

```
Dim ctlManyLabels(1 To 4) As Label
```

Les collections, approfondies à la section suivante, fonctionnent comme des groupes qui permettent d'accéder aux éléments comme on le fait avec un tableau. Au lieu de créer les objets lors de la conception, vous pouvez créer un groupe d'objets comme suit (remarquez l'utilisation du mot clé `New`) :

```
Dim fmrArray(1 To 10) As New frmFirstForm
```

L'instruction `Dim` considère qu'une feuille, `frmFirstForm`, existe. Après cette déclaration, il existe 10 nouvelles feuilles, dont les indices vont de `frmArray(1)` à `frmArray(10)`. On peut alors modifier par programme les propriétés des feuilles du groupe pour que chacune ait les particularités souhaitées, et les différencier de la feuille de base, `frmFirstForm`.

Chacune de ces feuilles ne s'affichera qu'à l'invocation de sa méthode `Show`.

Pour par exemple diminuer la taille de police des contrôles d'une feuille quand un utilisateur la redimensionne, vous pouvez utiliser la collection `Controls` pour le faire dans chaque contrôle :

```
1:  Private Sub Form_Resize ()
2:    ' Diminue la taille de police de tous les contrôles
3:    Dim intCtr As Integer
4:    For intCtr = 0 to Controls.Count - 1
5:      Controls(intCtr).FontSize = Controls(intCtr).FontSize * .75
6:    Next intCtr
7:  End Sub
```

La taille de police des contrôles sera réduite de 25 % par rapport à ce qu'elle était avant que l'utilisateur redimensionne la feuille.

Vous ne trouverez pas beaucoup de programmeurs Visual Basic qui utilisent des groupes de contrôles quand une collection existe pour le même objet (Visual Basic fournit une

collection `Forms`). Pour utiliser les groupes de contrôles, vous devez déclarer de la mémoire pour les contenir et pour initialiser les tableaux.

Visual Basic supporte une technique que vous utiliserez sans doute souvent pour créer les groupes de contrôles. Lorsque vous copiez un contrôle et que vous le collez à nouveau sur la feuille, Visual Basic affiche la boîte de message de la Figure 16.5.

Figure 16.5
Visual Basic va créer un groupe de contrôles pour vous.

Vous pouvez vous demander pourquoi copier et coller un contrôle, mais si vous avez à placer plusieurs boutons de commande ou étiquettes qui ont tous le même format, c'est une technique utile : vous créez le premier contrôle, configurez toutes ses propriétés, le copiez ensuite dans le Presse-papiers et enfin, vous le collez sur la feuille pour ajouter autant de contrôles que vous le voulez.

Dès que vous collez le contrôle, Visual Basic affiche la boîte de message de la Figure 16.5. Si vous répondez Oui, Visual Basic crée automatiquement un groupe de contrôles en utilisant le nom du premier. Par exemple, si le premier contrôle est `Command1`, les éléments commenceront à `Command1(0)` et l'indice augmentera tant que vous continuerez à coller le contrôle.

Votre code peut alors parcourir tous les éléments du groupe, de `Command1(0)` à `Command1(`*`n - 1`*`)`, *n* étant le nombre de contrôles du groupe, et définir des propriétés pour chacun.

Collections

Les collections jouent un rôle vital dans la programmation Visual Basic, comme vous avez déjà pu le voir. Elles sont toujours présentes et Visual Basic les met à jour automatiquement ; si vous ajoutez par exemple une feuille lors de l'exécution avec la déclaration `New Form`, Visual Basic actualise en conséquence la propriété `Count` de la collection `Forms`.

Les collections prédéfinies sont sans aucun doute utiles. Alors, pourquoi ne pas en créer une ? Visual Basic vous y autorise. Vous devrez cependant la gérer vous-même — ce qui demande plus d'effort que pour les collections prédéfinies.

Comme vous le savez maintenant, tous les objets appartiennent à une classe. Tout ce qui s'applique à une classe s'applique à ses objets. Par exemple, si un contrôle est un membre de la classe `CommandButton`, vous savez qu'il supporte l'événement Click, car tous les membres de la classe `CommandButton` supportent cet événement.

Vos collections propres doivent être des objets de la classe `Collection`. Elles se définissent au niveau du module en utilisant les mots clés `Public` ou `Private`, suivant la plage des procédures qui doivent y accéder. L'instruction suivante déclare une collection nommée `colMyCol` :

```
Private colMyCol As New Collection
```

Une collection fonctionne comme une étagère vide. Vous pouvez placer des objets (comme des livres), les ôter, les compter, etc. Naturellement, sur une étagère on peut mettre autre chose que des livres. Dans une collection, on ne peut mettre qu'un type d'élément, mais on peut déclarer plusieurs collections contenant chacune des types d'objets différents. Voici les méthodes auxquelles vos collections ont accès :

- `Add`. Ajoute un élément à la collection.
- `Count`. Renvoie le nombre d'éléments de la collection.
- `Item`. Sert d'index pour les éléments de la collection.
- `Remove`. Supprime un élément de la collection.

Comme le montre le Listing 16.3, Visual Basic s'occupe de la mise à jour de `Count` dans l'ajout d'éléments à la collection. Le code du Listing 16.3 crée une collection nommée `Cities` et y ajoute quatre éléments (des noms de ville).

Listing 16.3 : Utilisation de Add pour ajouter des éléments à la nouvelle collection

```
 1:  Dim Cities As New Collection
 2:  Dim intCtr As Integer
 3:
 4:  ' Ajoute les éléments
 5:  Cities.Add "Tulsa"
 6:  Cities.Add "Miami"
 7:  Cities.Add "New York"
 8:  Cities.Add "Seattle"
 9:
10: ' Montre qu'il y a quatre villes
11: frmMyForm.Print "Il y a "; Cities.Count; " villes :"
12:
13: ' Imprime chaque nom de ville
14: For intCtr = 1 To Cities.Count
15:   frmMyForm.Print "  "; Cities(intCtr)
16: Next
```

Si vous exécutez ce code, la sortie suivante s'affiche :

```
Il y a 4 villes :
Tulsa
Miami
New York
Seattle
```

Cette leçon ne fait qu'effleurer les capacités des collections. Vous devez cependant savoir que vous pouvez facilement y insérer et en ôter des éléments, dans l'ordre que vous préférez. Chaque élément de la collection est référencé par un indice, qui part de 1. Dans l'exemple précédent, `Cities(1)` est la première ville listée dans la collection. Les indices de collection commencent toujours à `1`, et non à `0` (comme le font les groupes de contrôles).

Vous pouvez utiliser l'*argument nommé* (un argument comprenant le nom de l'argument suivi de l'opérateur d'assignation, `:=`) `Before` pour ajouter des éléments à la collection à l'endroit où vous les voulez. La ligne suivante ajoute une ville au début de la collection `Cities`, quel qu'en soit le nombre d'éléments :

```
Cities.Add "St. Louis", Before:=1
```

Une position `Before` de 1 ajoute l'élément au début de la collection. Visual Basic ajoute l'élément avant l'indice spécifié. Si vous incluez cette instruction à la fin du code du Listing 16.3, vous obtiendrez la sortie suivante :

```
Il y a 5 villes :
St. Louis
Tulsa
Miami
New York
Seattle
```

Si vous n'aviez pas indiqué l'argument `Before:=1` dans l'instruction, St. Louis se retrouverait à la fin de la collection.

Vous pouvez supprimer des éléments spécifiques en utilisant la méthode `Remove`. Lorsque vous supprimez des éléments, les indices s'ajustent pour toujours commencer à 1. L'instruction suivante supprime le second élément (Tulsa) de la collection.

```
Cities.Remove 2
```

L'Explorateur d'objets

En progressant dans votre connaissance de Visual Basic, votre besoin d'outils performants augmente. Visual Basic en comporte un, l'Explorateur d'objets, qui vous permet d'inspecter les variables, les contrôles et d'autres objets de votre application. Les programmeurs Visual Basic qui découvrent l'Explorateur d'objets l'utilisent beaucoup plus souvent qu'ils ne pensent le faire, car ses caractéristiques simplifient beaucoup la programmation.

L'Explorateur d'objets vous aide à localiser et à gérer les objets de vos applications.

L'Explorateur d'objets est un référentiel en ligne complet, mais pas dans le même sens que l'aide en ligne. L'Explorateur d'objets vous offre un endroit unique où chercher les objets et leurs informations. Il vous permet aussi de sauter directement au code dont vous avez ensuite besoin.

L'Explorateur d'objets décrit les bibliothèques de types de votre application, qui sont des référentiels des informations de classes. Vous pouvez utiliser l'Explorateur d'objets pour accéder aux propriétés, méthodes et événements des objets de vos applications, y compris ceux que vous avez créés.

La fenêtre de l'Explorateur d'objets

Quand vous choisissez l'Explorateur d'objets dans le menu Affichage ou par son icône de la barre d'outils, sa fenêtre, illustrée à la Figure 16.6, s'affiche. Il se peut que vous ayez à l'agrandir et à fermer les fenêtres des propriétés et de la boîte à outils pour voir l'Explorateur d'objets dans son ensemble.

Le Tableau 16.2 décrit les parties de la fenêtre de l'Explorateur d'objets.

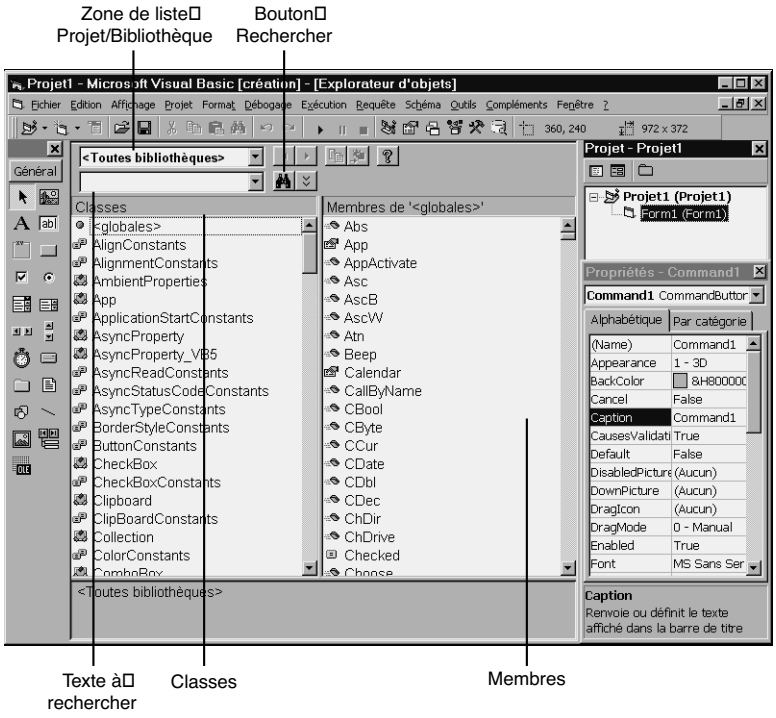


Figure 16.6
L'Explorateur d'objets décrit les objets de votre application.

Tableau 16.2 : La fenêtre de l'Explorateur d'objets comprend plusieurs éléments avec lesquels vous devez vous familiariser

Composant	*Description*
Zone de liste Projet/ Bibliothèque	Décrit la source des objets que vous voulez explorer. (Vous utiliserez souvent l'option `<Toutes bibliothèques>`, mais vous pouvez par exemple vous limiter aux objets propres à votre projet en sélectionnant son nom).
Texte à rechercher	Vous permet de saisir un objet, un événement, une méthode ou une propriété à rechercher.
Contrôles de manœuvre	Utilisés pour aller d'avant en arrière sur un chemin d'exploration déjà parcouru.
Classes	Contient les noms des classes du projet ou de la bibliothèque sélectionnée.
Membres	Contient les membres de la classe sélectionnée.

Parcourir l'Explorateur d'objets

L'Explorateur d'objets contient nombre d'informations identiques à celles fournies par le système d'aide en ligne. Mais il s'adresse plus particulièrement au programmeur Visual Basic et il présente les informations dont ce dernier a besoin de manière succincte. Par exemple, l'entrée <globales> de la liste des Classes décrit l'ensemble des fonctions intégrées à Visual Basic. Faites défiler la fenêtre des membres jusqu'à la rubrique `Left` pour voir des informations sur la fonction `Left()`.

Comme vous l'avez déjà étudié, la fonction `Left()` renvoie la partie de gauche d'une chaîne de caractères.

Lorsque vous sélectionnez l'entrée `Left`, Visual Basic décrit la fonction dans le bas de la fenêtre de l'Explorateur d'objets. Le texte décrit non seulement le but de la fonction, mais il montre aussi son format. Vous pouvez connaître la nature de chaque objet de la liste des Membres par son icône. Par exemple, la petite icône verte indique une fonction. Faites défiler la liste pour voir toutes les constantes nommées qui s'affichent après les fonctions et les collections dans la fenêtre Membres.

Si vous cliquez du bouton droit et sélectionnez Membres du groupe dans le menu contextuel, Visual Basic groupe les éléments Membres et Classes par usage, et non plus par ordre alphabétique.

Après avoir sélectionné une entrée dans la fenêtre de l'Explorateur d'objets, cliquez sur le bouton Aide de la barre d'outils (icône avec un point d'interrogation) pour obtenir les informations de l'aide en ligne le concernant.

Vous pouvez être encore plus spécifique avec l'Explorateur d'objets. La liste des Classes contient entre autres plusieurs entrées qui référencent des constantes nommées. Quand vous cliquez sur l'entrée `ColorConstants`, par exemple, seules les constantes nommées de couleurs de Visual Basic s'affichent dans la liste des membres (voyez la Figure 16.7).

Vous pouvez utiliser ces constantes nommées partout dans votre code où vous avez besoin d'une couleur. Par exemple, vous pouvez configurer la couleur d'arrière-plan d'une feuille ainsi :

```
frmMyForm.BackColor = vbRed     ' Configure la feuille en rouge
```

Tous les contrôles de la boîte à outils apparaissent également dans la liste des Classes. Si vous cliquez sur ComboBox, par exemple, l'Explorateur d'objets affiche toutes les informations pertinentes le concernant, ses propriétés, ses méthodes et ses événements.

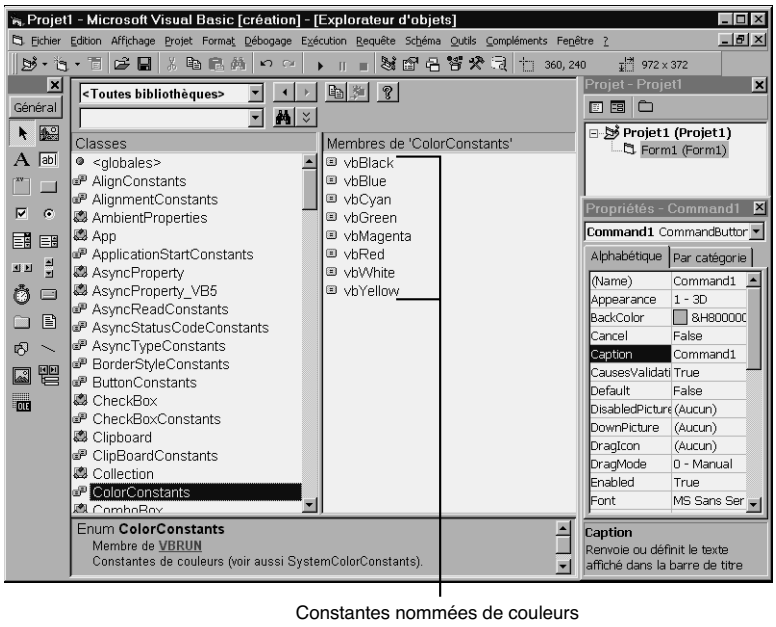


Figure 16.7
La recherche des objets a été réduite à des constantes particulières.

Si vous cliquez sur une des entrées dans la liste des Membres, vous en obtiendrez une description.

Les programmeurs utilisent l'Explorateur d'objets pour de nombreux usages. Il affiche les objets d'une manière organisée. Outre les objets, il coordonne toutes les particularités de votre programme. Par exemple, si vous écrivez du code et voulez utiliser des fonctions internes de date et d'heure, cliquez sur l'entrée DateTime dans la liste de Classes. Comme le montre la Figure 16.8, la liste des Membres se met à jour et ne montre que les fonctions intégrées qui concernent les dates et heures.

Même si ces fonctions sont également listées dans l'aide en ligne ou ailleurs, pouvoir réviser toutes les fonctions concernant la date et l'heure simplifie la programmation, car elles se retrouvent classées thématiquement.

Le bouton Précédent retrace votre cheminement dans l'Explorateur d'objets. Il est donc facile d'aller d'avant en arrière avec la souris. Cela imite le comportement des navigateurs Web utilisés actuellement.

L'Explorateur d'objets est aussi très utile pour décrire votre projet. Lorsque vous ajoutez des objets, des variables et des procédures événementielles, l'Explorateur d'objets s'occupe dans l'ombre à tout recenser. La Figure 16.9 montre le `Projet1` sélectionné

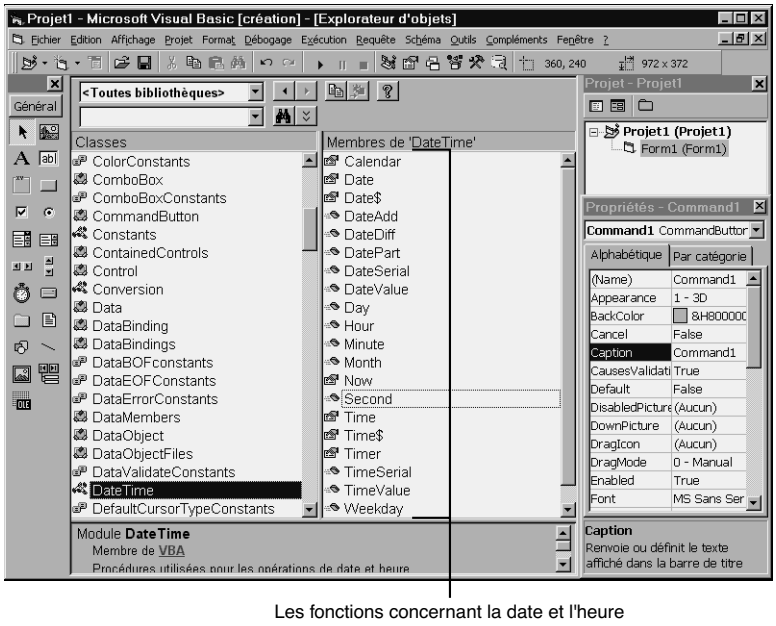


Figure 16.8
Trouver la fonction intégrée sur un certain sujet est facile.

dans la zone de liste Projet/Bibliothèque. Quand vous cliquez sur `Form1` (pour l'instant le seul objet de ce projet), l'Explorateur d'objets affiche la liste complète des procédures événementielles, des méthodes et des propriétés correspondantes. Une seule entrée, `Command1_Click` est en gras, ce qui signifie que du code a été ajouté dans cette procédure.

Un des aspects les plus puissants de l'Explorateur d'objets est l'option Afficher la définition. Si vous sélectionnez un membre qui fait partie de vos propres objets (par exemple une procédure événementielle que vous avez écrite ou un objet que vous avez déclaré), puis que vous cliquez du bouton droit et sélectionnez Afficher la définition, Visual Basic saute directement au code où cet objet est défini. Vous pouvez ainsi utiliser l'Explorateur d'objets pour localiser du code spécifique dans une application volumineuse. Vous n'avez pas besoin de savoir dans quel projet l'objet a été défini tant que vous pouvez le localiser dans l'Explorateur d'objets.

Quand vous cherchez un élément dans l'Explorateur d'objets, vous obtenez une liste de toutes les références faites à cet élément dans votre application. Par exemple, si vous cherchez `Click`, Visual Basic affiche une fenêtre supplémentaire (voyez la Figure 16.10) qui contient toutes les références en rapport dans le projet (et dans tout le répertoire Visual Basic, car l'option `<Toutes bibliothèques>` est sélectionnée).

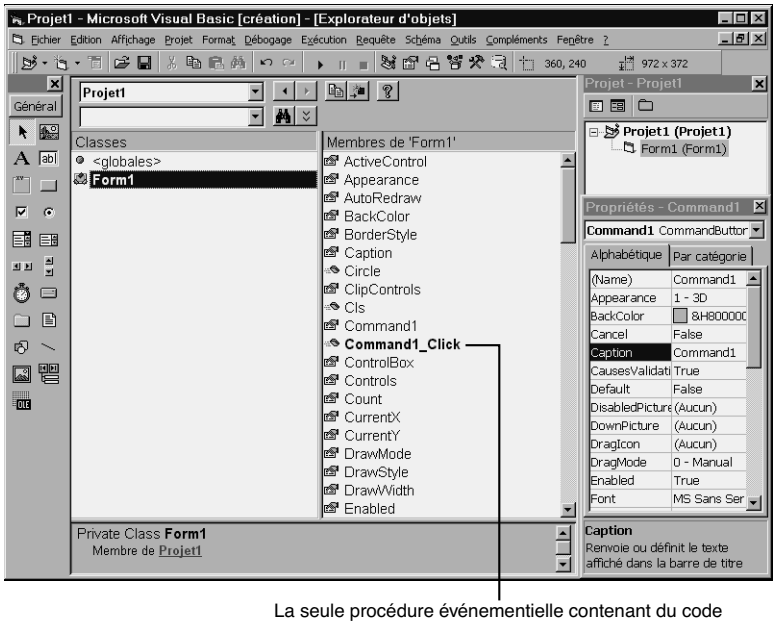


Figure 16.9
L'Explorateur d'objets ne montre que les caractéristiques spécifiques de l'application.

L'événement `Click` se produit sur divers objets, donc l'Explorateur d'objets affiche une fenêtre qui contient tous les objets référencés qui supportent `Click`. Vous pouvez ensuite affiner la recherche en cliquant sur l'objet ou le projet qui contient l'événement `Click` que vous recherchez.

En résumé

Ce chapitre n'a traité que des objets. Vous avez appris la manière dont les objets OLE fonctionnaient dans Visual Basic. Le contrôle OLE ne fait rien en lui-même si ce n'est contenir des objets (ou des liens vers des objets) d'autres applications. Une fois ces objets en place, l'utilisateur peut les modifier et les manipuler à l'intérieur de votre application. Vous n'avez pas à écrire de commandes pour accéder à l'autre application, car les objets OLE sont intelligents et ils en amènent tous les outils d'édition, y compris les menus.

Outre les collections, Visual Basic vous permet de déclarer des groupes de contrôles, qui fonctionnent souvent comme des tableaux d'objets. Vous pouvez dupliquer les contrôles

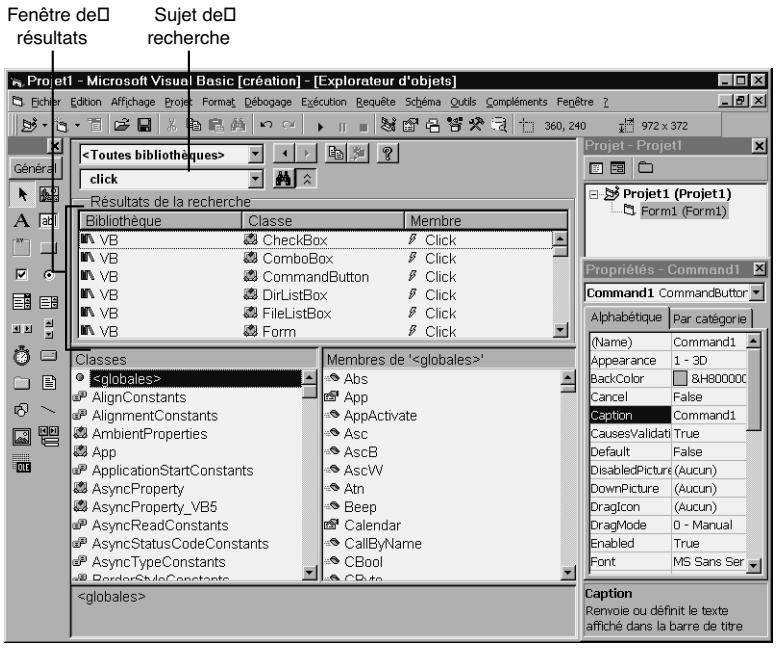


Figure 16.10 *La fonction de recherche a trouvé toutes les occurrences de* `Click`.

d'une feuille pour qu'ils partagent des caractéristiques communes, en n'ayant plus qu'à modifier les propriétés qui diffèrent (par exemple le titre ou la couleur). Les objets système proposent des objets prédéfinis avec lesquels votre application peut interagir avec les ressources extérieures à l'environnement classique du programme. En accédant à l'objet `App`, votre application peut déterminer en cours d'exécution le chemin à partir duquel elle a été lancée. L'objet `Clipboard` permet à votre application d'interagir avec le Presse-papiers de Windows pour copier et coller du texte.

Visual Basic fournit un outil d'organisation, l'Explorateur d'objets, qui est essentiellement un référentiel de données. Vous pouvez rechercher des événements, des méthodes ou des propriétés spécifiques ou consulter toute une classe d'objets. L'Explorateur d'objets suit même dans votre code les objets que vous avez initialisés et les procédures événementielles que vous avez écrites.

La leçon de demain passe à la génération suivante des objets en décrivant comment utiliser et créer les contrôles ActiveX.

Questions-réponses

Q L'utilisateur doit-il disposer de l'application d'origine de l'objet OLE pour que ce dernier fonctionne dans l'application Visual Basic qui l'utilise ?

R Oui, car l'automatisation in situ exige les menus et toutes les caractéristiques de l'application d'origine. Si cette dernière n'est pas installée, l'utilisateur sera incapable de modifier l'objet OLE. Microsoft Excel, par exemple, tableur très puissant qui occupe beaucoup d'espace disque et de mémoire, est compatible OLE. Vous pouvez donc amener une feuille Excel dans une application que vous concevez, et l'utilisateur disposera alors de toute sa puissance dans votre programme. Etant donné la taille d'Excel, de ses menus et de ses caractéristiques, il ne pourrait jamais être intégré dans votre application avec son objet. Il faut donc qu'Excel soit installé pour pouvoir utiliser un objet Excel dans votre application. Vous devez clairement déclarer les exigences de votre application, dont la présence de programmes auxiliaires tels que Microsoft Excel, pour que l'on puisse l'utiliser pleinement.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités et des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se trouvent à l'Annexe A.

Quiz

1. Quelle est la différence entre liaison et incorporation ?
2. Quel type de technique OLE, liaison ou incorporation, consomme à votre avis le plus d'espace disque ?
3. Vrai ou faux. Visual Basic enregistre automatiquement les modifications que fait l'utilisateur aux objets OLE incorporés.
4. Quelle est la méthode permettant d'enregistrer un objet OLE dans un fichier disque ?
5. Quelle est la méthode permettant de lire un objet OLE à partir d'un fichier disque ?
6. Indiquez deux manières de tester la classe d'un objet.

7. Vrai ou faux. Vous devez transmettre les objets système si vous devez y accéder à partir de plusieurs procédures.
8. Indiquez trois types d'éléments qui apparaissent souvent dans la liste des Membres de l'Explorateur d'objets.
9. Que se passe-t-il si vous groupez la liste des Membres dans l'Explorateur d'objets ?
10. Vrai ou faux. L'Explorateur d'objets ne recherche pas les objets que vous avez créés.

Exercices

1. Pourquoi l'utilisation de `With… End With` est-elle sans doute une mauvaise idée dans ce cas ?

```
With chkMaster
  .Caption = "Source principale"
  .Alignment = vbLeftJustify
End With
```

2. Ecrivez une application simple qui contient un contrôle OLE. Incorporez-y l'objet Image Paintbrush de Windows. Ajoutez une option de menu permettant d'enregistrer les images et une autre pour les charger. Exécutez l'application, double-cliquez sur le contrôle et dessinez une image. Enregistrez-la, quittez l'application, puis redémarrez-la. Chargez l'image enregistrée pour vous assurer que les procédures de chargement et d'enregistrement sont correctes.

Chapitre 17

Contrôles ActiveX

La leçon d'aujourd'hui vous montre comment appréhender et utiliser les contrôles ActiveX. En ajoutant des contrôles ActiveX à votre fenêtre de boîte à outils, vous ajoutez des fonctionnalités à l'environnement Visual Basic et réduisez le temps nécessaire au développement des applications.

La technologie objet supportée par Visual Basic vous permet d'emprunter des fonctionnalités à d'autres applications qui supportent ActiveX et l'automatisation ActiveX. Par exemple, votre application Visual Basic peut créer un document Word ou une feuille de calcul Excel en empruntant la technologie de ces applications externes.

Une fois que vous aurez compris l'importance de la technologie des contrôles ActiveX et les nombreuses manières dont vous pouvez en tirer parti, vous apprendrez comment créer vos propres contrôles ActiveX.

Aujourd'hui vous apprendrez :

- l'historique des contrôles ActiveX ;
- le rapport entre contrôles VBX et OCX et contrôles ActiveX ;
- comment ajouter des contrôles ActiveX à vos projets ;
- l'automatisation des objets dans vos applications ;
- comment créer une feuille de calcul Excel à partir d'une application Visual Basic ;
- comment créer de nouveaux contrôles ActiveX ;
- le sous-classement des nouveaux objets ActiveX ;
- comment créer des valeurs de propriétés propres à vos contrôles.

La technologie ActiveX

La technologie ActiveX est la version actuelle de l'ensemble de contrôles d'extension VBX apparus il y a quelques années. L'extension de fichier .VBX signifie "Visual Basic eXtended". En d'autres termes, les contrôles ActiveX sont les descendants des contrôles que vous pouviez ajouter aux versions antérieures de Visual Basic pour étendre la boîte à outils livrée avec le produit.

Dans le passé, les contrôles Visual Basic n'étaient pas compatibles avec la technologie des navigateurs Web, l'Internet et les autres outils de programmation de Windows, tels que Visual C++. Cependant, ils étaient importants pour permettre aux programmeurs d'étendre leur capacité à écrire du code. Plus le programmeur dispose de contrôles, moins il a de travail à faire. En conséquence, une communauté et toute une activité se sont développées pour créer des contrôles Visual Basic manipulant des graphiques, des données, des grilles, du multimédia, etc. Leur succès a obligé Microsoft à les reconcevoir (car ils n'étaient compatibles avec aucun autre produit Microsoft).

OLE est une technique cousine des contrôles VBX, mais qui est plus universelle dans les applications Microsoft — elle n'a pas été conçue exclusivement pour les programmeurs Visual Basic. Cependant, comme vous avez pu le voir hier, un objet OLE est un objet de donnée (et pas un contrôle) utilisable comme un contrôle Visual Basic.

Entre les contrôles Visual Basic et ActiveX, Microsoft a conçu des contrôles OCX 32 bits particuliers. Ces nouveaux contrôles Visual Basic étendaient les capacités de Visual Basic, mais aussi d'autres langages de programmation comme Visual C++. Les contrôles OCX avaient une extension de fichier .OCX. Les anciens contrôles VBX ne reconnaissent que les applications 16 bits.

Si vous avez travaillé avec de précédentes éditions de Visual Basic qui supportaient des contrôles VBX 16 bits, ce qui a été le cas jusqu'à la version 5.0, le système actuel ne peut les utiliser que s'il dispose de nouvelles versions 32 bits. Par exemple, Visual Basic est fourni avec une version 32 bits du contrôle Gauge (la version 16 bits faisait partie des versions antérieures de Visual Basic). Donc, si vous chargez une ancienne application Visual Basic qui utilise ce contrôle, Visual Basic le remplacera par la version 32 bits, et tout devrait fonctionner correctement. Si, par contre, un remplacement n'existe pas ou qu'il ne peut pas être fourni par le distributeur du contrôle VBX, vous devrez supprimer le contrôle de l'application et lui substituer un contrôle identique.

Les contrôles OCX, bien que compatibles avec Visual C++, ne fonctionnaient pas simplement sur l'Internet ; Microsoft les a donc fait évoluer en contrôles ActiveX, pour permettre aux navigateurs Internet et aux multiples applications et langages de programmation de fonctionner correctement avec eux.

Y aura-t-il un remplacement à ActiveX ? Sans doute un jour, suivant les nouvelles techniques qui demanderont des fonctionnalités que ne propose pas la technologie ActiveX.

Les versions actuelles d'Internet Explorer et de Netscape (à l'aide d'un module d'extension disponible sur la page de support de Netscape), supportent ActiveX sur les pages Web. Ce qui signifie que les utilisateurs peuvent interagir avec les contrôles ActiveX dans toutes les pages Web qui en contiennent.

Ajout de contrôles ActiveX à un projet

Les contrôles ActiveX (pour vous, programmeur Visual Basic) ne représentent que des contrôles additionnels qui peuvent être ajoutés à la fenêtre Boîte à outils et utilisés pour développer des programmes. Tous types de contrôles existent. Plusieurs sont fournis avec Visual Basic ; vous pouvez les trouver en sélectionnant Projet, Composants (ou Ctrl-T), comme vous l'avez déjà fait. En outre, vous trouverez des contrôles ActiveX sur l'Internet. Des entreprises de programmation vendent également des contrôles ActiveX qui peuvent être fusionnés dans l'environnement Visual Basic.

*Consultez le site Web de Microsoft (**http://www.microsoft.com/activex**) pour avoir des exemples de contrôles ActiveX que vous pouvez télécharger sur votre PC.*

N'ajoutez à la fenêtre Boîte à outils que les contrôles nécessaires à votre application. Chaque contrôle supplémentaire est envoyé dans l'application compilée que vous distribuez. Plus l'application contient de contrôles, plus elle sera volumineuse, plus elle s'exécutera lentement et plus elle consommera de ressources sur la machine de l'utilisateur. Pour supprimer les contrôles ActiveX inutiles, affichez la boîte de dialogue Projet, Composants et décochez tous les contrôles qui n'appartiennent pas à votre projet.

Qu'une application utilise tous les contrôles ActiveX chargés au moment de la compilation ou pas, il est obligatoire de distribuer les fichiers exécutables ActiveX avec le projet.

Un fichier exécutable de contrôle ActiveX est un fichier annexe que vous devez fournir avec l'application qui utilise le contrôle ActiveX correspondant. Ce fichier contient les instructions nécessaires pour que le contrôle ActiveX fonctionne lors de l'exécution.

Il existe des contrôles ActiveX pour pratiquement toutes les tâches de programmation que vous pouvez avoir à faire. Pour un meilleur contrôle du son que celui offert par le contrôle multimédia, vous pourrez trouver de nombreux types de contrôles ActiveX relatifs au son en ligne. De même, les contrôles graphiques 3D, les contrôles Internet, les contrôles mathématiques, d'imprimantes, de scanners et un vaste assortiment d'autres contrôles sont disponibles. Vous devrez décider du type de programmation que vous effectuez le plus souvent et rechercher les contrôles particuliers qui peuvent vous assister. Par exemple, si vous écrivez un logiciel de publication, vous chercherez tous les contrôles de manipulation et d'édition de texte possibles.

Lorsque vous ajoutez un nouveau contrôle ActiveX à la fenêtre Boîte à outils, comment l'utiliser ? Pour commencer, vous pouvez considérer que les contrôles ActiveX supportent les éléments suivants :

- Propriétés ;
- Evénements ;
- Méthodes.

Autrement dit, vous utilisez un contrôle ActiveX exactement comme n'importe quel autre contrôle. Il vous faudra obtenir une liste des propriétés, événements et méthodes supportés par le contrôle pour pouvoir le programmer correctement. Si de nombreuses propriétés des contrôles apparaissent parfois dans la fenêtre Propriétés, ce n'est pas toujours le cas (en particulier pour les contrôles disponibles à l'exécution). Lorsque vous achetez des contrôles ActiveX ou que vous les téléchargez depuis les sites Internet qui les proposent, vous devez aussi obtenir des instructions qui listent les propriétés, événements et méthodes. Les contrôles ActiveX fournis avec Visual Basic sont référencés dans l'aide en ligne.

La Figure 17.1 montre la boîte de dialogue Composants, qui s'affiche quand vous sélectionnez Projet, Composants. Vous avez déjà rencontré cette boîte de dialogue plusieurs fois dans ce didacticiel, car vous avez déjà ajouté des contrôles ActiveX auparavant, par exemple le contrôle multimédia.

Lorsque vous installez Visual Basic, plusieurs contrôles ActiveX sont ajoutés à votre système ; vous pouvez à votre tour les ajouter à un projet par la boîte de dialogue Composants. De plus, la routine d'installation de Visual Basic recherche aussi sur votre système des contrôles supplémentaires à ajouter, dont nombre apparaîtront dans la boîte de dialogue Composants. Une installation de Visual Basic sur deux ordinateurs

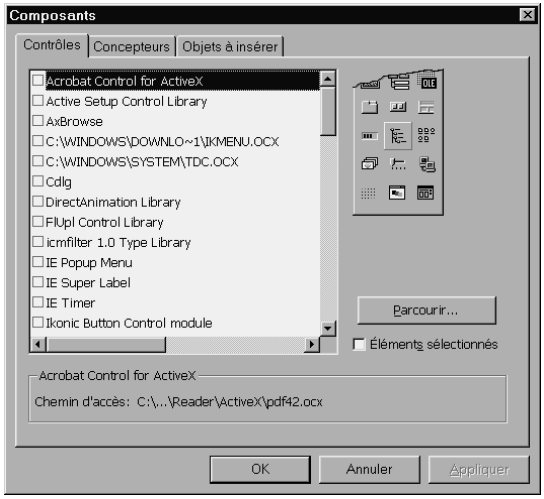


Figure 17.1
Utilisez la boîte de dialogue Composants pour ajouter de nouveaux contrôles ActiveX à votre projet.

différents peut donc présenter des ensembles de contrôles ActiveX différents. Cliquez sur le bouton Parcourir de la boîte de dialogue pour rechercher les fichiers de contrôles ActiveX de votre disque dur.

La liste des contrôles ActiveX de la boîte de dialogue Composants peut devenir assez longue. Vous pouvez cliquer sur l'option Eléments sélectionnés pour ne voir apparaître dans la liste que les contrôles ActiveX que vous utilisez. Cependant, pour ajouter d'autres éléments, vous devrez décocher l'option pour pouvoir trouver le contrôle que vous voulez ajouter.

Lorsque vous ajoutez des contrôles dans la fenêtre Boîte à outils, elle peut vite se remplir. Observez l'onglet Général au sommet de la boîte à outils. En cliquant du bouton droit sur une zone vierge dans la fenêtre Boîte à outils, vous pouvez y créer de nouveaux onglets pour regrouper des contrôles ActiveX. Pour ajouter des contrôles dans un nouveau groupe, cliquez sur l'onglet correspondant, puis ajoutez-les. (La boîte de dialogue Composants est disponible à partir du menu contextuel de la fenêtre Boîte à outils qui s'affiche par un clic droit.) La Figure 17.2 montre un ensemble de contrôles ayant rapport à Internet ajoutés dans un groupe conçu dans ce but.

Figure 17.2
Les onglets de groupes permettent d'organiser la fenêtre Boîte à outils.

Automatisation ActiveX

Jusqu'ici, vous avez ajouté quelques contrôles et vous savez tout du paramétrage des propriétés, de la réaction aux événements et du déclenchement des méthodes. Certains contrôles ActiveX vous permettent cependant d'aller un cran au-delà. Vous pouvez en fait utiliser dans votre application un contrôle incorporé et en emprunter les fonctionnalités.

Visual Basic supporte une telle *automatisation* des contrôles entre applications. Par exemple, vous pouvez ouvrir Excel, charger une feuille de calcul, en manipuler les données à partir de commandes Excel, fermer Excel, puis incorporer la feuille de calcul résultante dans la fenêtre de votre application sans que les utilisateurs se doutent un instant que vous avez emprunté les caractéristiques d'Excel.

L'automatisation est le traitement par lequel une application utilise les données d'une autre application et les manipule avec l'aide de cette dernière. Les utilisateurs ne se rendent pas compte que l'autre application a été lancée, a travaillé et a été refermée.

L'utilisation de l'automatisation se limite aux applications ActiveX enregistrées dans la base de registres de votre machine. En général, si vous utilisez une application compatible ActiveX, elle a enregistré sa capacité d'automatisation dans la base de registres lors de son installation.

L'automatisation exige normalement une connaissance approfondie de la hiérarchie des objets de l'autre application, qui peut être complexe. L'exemple qui suit vous apprend l'automatisation par une démonstration utilisant un document ActiveX. Pour comprendre pleinement l'automatisation, vous devez être versé dans les fonctionnalités internes de l'application empruntée. Cet ouvrage ne peut pas entrer dans les particularités des applications autres que Visual Basic. Heureusement, la plupart des concepts

de l'automatisation dépassent le cadre d'une application unique. Les concepts de l'exemple que vous verrez ici se transposent largement à d'autres applications.

Pour commencer cet exemple, vous devez définir une variable qui représente l'application à automatiser dans votre propre programme. Vous utiliserez le type de données `Object` pour créer une variable référençant l'application d'automatisation. Vous devez d'abord définir un objet application ainsi :

```
Dim obExcelApp As Object
```

Vous devez ensuite connecter la variable à l'application. Si cette dernière n'est pas en cours d'exécution, vous devez la lancer à l'arrière-plan par la fonction `CreateObject()`. Cette fonction ne se limite pas à démarrer l'application, mais elle y connecte aussi votre variable objet, ainsi :

```
Set obExcelApp = CreateObject("Excel.Application")
```

L'argument de la fonction `CreateObject()` est le nom de l'application.

Utilisez `Set` au lieu d'une simple assignation pour attacher l'application automatisée à votre application Visual Basic. Une variable ne peut pas contenir une application externe ; elle ne peut contenir que des valeurs telles que des nombres et des chaînes de caractères.

Utilisez `Set` pour créer une référence à l'objet externe. `Set` *n'assigne pas, mais fait pointer la variable sur l'objet qu'elle représente.*

Un problème peut se poser si l'application s'exécute déjà. Dans un système multitâche et multi-utilisateur, plusieurs copies d'Excel peuvent s'exécuter au même moment. Vous pouvez utiliser la fonction `GetObject()` à la place de `CreateObject()` si l'application est en cours d'exécution :

```
Set obExcelApp = GetObject(, "Excel.Application")
```

Remarquez la virgule au début de la liste des arguments. Vous pouvez, dans la plupart des cas, omettre le premier argument, car le second décrit l'objet que vous obtenez. Si le second argument est absent, vous devez fournir, comme premier argument, le chemin d'accès à un fichier qui décrit l'objet que vous voulez créer.

Si Excel est déjà en cours d'exécution, il est préférable de ne pas en lancer une seconde instance. En utilisant le déroutement des erreurs, vous pouvez vérifier si c'est le cas. La fonction `GetObject()` déclenchera une erreur si Excel n'est pas en cours d'exécution.

Dans ce cas, vous pouvez alors utiliser `CreateObject()` pour démarrer une instance d'Excel.

Voici un exemple de code que vous pouvez utiliser pour vérifier qu'une instance d'Excel est démarrée :

```
 1:  ' Déroutement des erreurs
 2:  On Error Resume Next
 3:  '
 4:  ' Référence l'application Excel
 5:  Set obExcelApp = GetObject(, "Excel.Application")
 6:  If Err.Number <> 0 Then
 7:     Set obExcelApp = CreateObject("Excel.Application")
 8:     blnRunning = False        ' Excel n'était pas en exécution
 9:  Else
10:    blnRunning = True
11: End If
```

Vous avez déjà étudié l'instruction `On Error Goto` lors de leçons précédentes, mais c'est la première fois que vous rencontrez l'option `Next`. Jusqu'ici, l'instruction `On Error` se déroutait en cas d'erreur sur une étiquette d'instruction. L'option `Next` indique simplement à Visual Basic, quand il rencontre une erreur, de passer à l'instruction suivante et de continuer le programme. Cette situation fournit l'occasion de dire qu'un code d'erreur est renvoyé chaque fois que l'instruction `On Error` détecte une erreur. Ce code est une propriété d'un objet prédéfini du système nommé `Err`.

Tant qu'il n'y a pas d'erreurs dans un programme en exécution, `Err.Number` est à `0`. Par conséquent, si `Err.Number` contient une valeur différente, cela signifie qu'une erreur s'est produite. Dans l'exemple, la ligne 6 signalera un code d'erreur dans `Err.Number` si la fonction `GetObject()` échoue. Donc, la ligne 7, déduisant qu'Excel n'est pas déjà démarré, lance une instance d'Excel par la fonction `CreateObject()`. (Pour détecter d'autres erreurs plus loin dans le programme, comme vous devriez le faire si vous affichez des boîtes de dialogue et que vous aviez besoin de savoir si l'utilisateur a cliqué sur Annuler, vous pouvez réinitialiser l'état de l'erreur en mettant `Err.Number` à `0`.) La variable booléenne `blnRunning` est mise à `False` pour que le programme sache qu'il a lancé Excel.

Si Excel était déjà en cours d'exécution, il ne faut pas l'arrêter dans votre programme.

Quand vous ouvrez une autre application en utilisant l'automatisation, votre application doit comprendre dans le détail son interface. D'une certaine manière, votre programme est un utilisateur de l'autre application. Donc, si vous ouvrez Excel, vous interagissez avec lui en utilisant la notation classique de ligne et de colonnes, mais vous devez aussi utiliser une notation de propriétés des objets spécifique à Excel.

Les variables d'objets d'automatisation d'application sont une exception à la règle générale qui dit qu'il est préférable de n'utiliser que des variables locales. L'opération étant réellement en dehors de votre application, vous pouvez utiliser en toute sécurité une variable objet globale pour que vos procédures n'aient pas à transmettre la variable de l'application.

Vous devez maintenant déclarer un objet feuille de calcul pour que l'application puisse générer des données :

```
Dim obWorkSheet As Object  ' Objet feuille de données
```

Le code suivant ajoute des données à certaines cellules de la feuille de données :

```
' Entrer des valeurs dans des cellules
obWorkSheet.Cells(1, 1).Value = "Ventes"
obWorkSheet.Cells(1, 2).Value = "Mois"
obWorkSheet.Cells(2, 1).Value = 21913.44
obWorkSheet.Cells(2, 2).Value = "avril"
```

Si vous assemblez les éléments de la section précédente et ajoutez un peu de code de nettoyage, comme le code d'enregistrement de la feuille de calcul et de fermeture de l'objet Excel, vous devriez aboutir à quelque chose qui ressemble au Listing 17.1.

Listing 17.1 : Votre application peut utiliser Excel pour créer une feuille de calcul

```
 1:  Private Sub cmdSendToExcel_Click()
 2:    Dim obExcelApp As Object     ' Objet Appplication
 3:    Dim obWorkSheet As Object    ' Objet Feuille de calcul
 4:    Dim blnRunning As Boolean    ' Si Excel était en exécution
 5:
 6:    ' Déroutement des erreurs
 7:    On Error Resume Next
 8:    '
 9:    ' Référencer l'application Excel
10:   Set obExcelApp = GetObject(, "Excel.Application")
11:    If Err.Number <> 0 Then
12:       Set obExcelApp = CreateObject("Excel.Application")
13:       blnRunning = False        ' Excel n'était pas en exécution
14:    Else
15:       blnRunning = True
16:    End If
17:
18:    ' Ajouter un nouveau classeur
19:    obExcelApp.Workbooks.Add
20:
21:    ' Référencer la feuille de calcul active
22:    Set obWorkSheet = obExcelApp.ActiveSheet
23:
```

Listing 17.1 : Votre application peut utiliser Excel pour créer une feuille de calcul *(suite)*

```
24:    ' Entrer des valeurs dans les cellules de la feuille active
25:    obWorkSheet.Cells(1, 1).Value = "Ventes"
26:    obWorkSheet.Cells(1, 2).Value = "Mois"
27:    obWorkSheet.Cells(2, 1).Value = 21913.44
28:    obWorkSheet.Cells(2, 2).Value = "avril"
29:
30:   ' Sélectionner la deuxième ligne pour formater
31:    obWorkSheet.Rows("2:2").Select
32:    obExcelApp.Selection.NumberFormat = "$##,###.##"
33:
34:    ' Enregistrer le classeur (changez ce nom s'il existe déjà)
35:    obExcelApp.Save ("c:\VBCreated.XLS")
36:
37:    ' Ne pas quitter si Excel était déjà lancé !
38:    obExcelApp.ActiveWorkBook.Close False
39:
40:   If Not (blnRunning) Then ' S'il n'était pas lancé...
41:     obExcelApp.Quit        ' alors quitter Excel
42:   End If
43:
44: End Sub
```

Si vous avez vérifié qu'Excel n'était pas déjà en cours d'exécution, vous pouvez le fermer (comme le fait la ligne 41). Dans le cas contraire (la fonction `GetObject()` n'ayant pas renvoyé d'erreur), il n'est pas souhaitable de quitter Excel, car il peut être en train de s'exécuter en tâche de fond. Le code de cet exemple crée la petite feuille de calcul illustrée à la Figure 17.3.

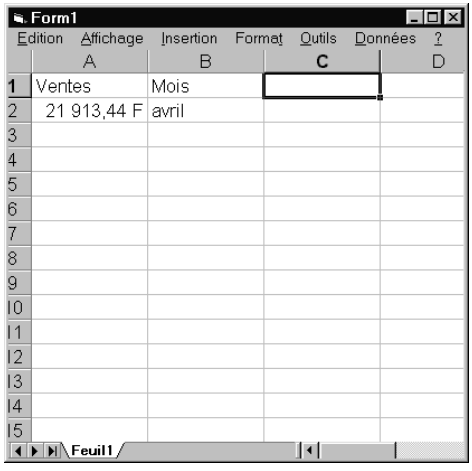


Figure 17.3
Votre application Visual Basic peut créer des feuilles de calcul Excel !

Cette feuille de calcul est simple pour permettre à l'exemple d'avoir une taille raisonnable. D'ordinaire, votre application Visual Basic peut modifier des valeurs et même déclencher un graphique Excel et imprimer un état. La chose importante à retenir est que Visual Basic a utilisé l'intelligence d'Excel pour créer une feuille de calcul mise en forme sans que l'utilisateur, à son clavier, ne sache qu'Excel est impliqué.

Une fois que vous avez utilisé Excel ou Word ou une autre application compatible ActiveX, vous devez inclure cet objet dans votre application Visual Basic.

Excel contient son propre langage d'automatisation, comme Word et toutes les applications compatibles avec l'automatisation ActiveX. Cependant, la plupart des applications supportent les caractéristiques d'ouverture et de fermeture présentées dans cette section pour connecter une application et son objet de données principal à des variables objet Visual Basic. De même, elles supportent des méthodes et des propriétés comme celles qui sont illustrées ici. Vous n'aurez que peu de problèmes tant que vous comprenez Visual Basic. Vous devez cependant avoir accès au langage interne de l'application utilisée pour l'automatisation. Recherchez dans son aide en ligne pour trouver sa hiérarchie des objets.

Création de vos propres contrôles ActiveX

Si l'idée de créer un contrôle ActiveX vous effraie un peu, considérez les raisons qui peuvent vous amener à le faire. Vous pourriez non seulement distribuer (ou même vendre) vos contrôles à d'autres développeurs, mais aussi les réutiliser dans vos propres applications. Par exemple, si vous vous retrouvez à toujours refaire les mêmes types de modifications aux contrôles de Visual Basic pour qu'ils fonctionnent comme vous le voulez, écrivez de nouveaux contrôles qui ne se contentent pas d'imiter ceux dont vous disposez, mais qui présentent des propriétés et des méthodes intégrées dont vous avez besoin ! La prochaine fois que vous avez à écrire une application qui a besoin de ce contrôle, vous n'avez qu'à l'ajouter à la boîte à outils et définir les propriétés nécessaires. Au lieu d'avoir à récrire le même code pour que le contrôle se comporte d'une certaine manière, il suffit de paramétrer les valeurs de ses propriétés et de revenir aux détails importants de l'application.

Un autre avantage lié au développement de votre propre collection d'outils de programmation ActiveX est la possibilité de porter ces outils vers d'autres langages de programmation qui supportent ActiveX, par exemple Visual C++.

Concevoir les contrôles

Visual Basic contient des outils qui vous aident à concevoir vos propres contrôles ActiveX, même si le processus doit se passer en plusieurs étapes. Pour illustrer ce que cela implique, le reste du Chapitre vous guide dans la conception d'un nouveau type de zone de texte.

Détails de la nouvelle zone de texte

La nouvelle zone de texte que vous allez créer étendra le contrôle TextBox générique de Visual Basic en proposant les caractéristiques suivantes :

- Il reconnaîtra toutes les valeurs classiques des propriétés reconnues par le contrôle TextBox standard.
- Il contiendra aussi une nouvelle propriété, AutoTSize qui reconnaît quatre valeurs : `1-NA`, `2-Small`, `3-Medium` et `4-Large`. Ces valeurs apparaîtront comme une énumération dans une liste déroulante de la fenêtre Propriétés. Vous pouvez assigner, dans votre code, 1, 2, 3 ou 4 à la zone de texte pour paramétrer la valeur. La valeur `1-NA`, qui est la valeur par défaut, ne modifie pas la taille courante de police de la zone de texte, définie dans la propriété `Font.Size`. Lorsqu'on attribue la valeur `2-Small`, le texte de la zone de texte sera dimensionné à 25 % de la valeur `Height`. La valeur `3-Medium` met la taille du texte à la moitié de la valeur `Height`. Enfin, la valeur `4-Large` mettra le texte à 75 %. (La valeur de la propriété `Font.Size` se modifiera pour refléter la nouvelle taille.) Cela donne une manière simple de paramétrer le texte de la zone de texte à une de ces trois valeurs.

Une énumération est une liste de valeurs fixes pouvant être prises par un contrôle, par exemple, `True` et `False`. Les valeurs énumérées apparaissent dans une liste déroulante en face de la propriété dans la fenêtre Propriétés.

- Il contiendra également deux nouvelles propriétés appelées `UCase` et `LCase`, qui seront des propriétés booléennes. Si `UCase` est positionné à `True`, le texte de la zone de texte sera converti en majuscules. Si `LCase` est positionné à `True`, le texte sera converti en minuscules. `UCase` et `LCase` sont par défaut tous deux paramétrés à `False` ; les deux ne pouvant être `True` en même temps. Votre contrôle doit donc s'assurer que, lorsqu'une de ces propriétés est à `True`, l'autre se positionne à `False`.

Le nouveau contrôle ActiveX que vous créerez ressemblera et fonctionnera comme les autres. Vous pourrez l'insérer dans la fenêtre Boîte à outils, double-cliquer dessus pour l'ajouter à la fenêtre feuille, et sélectionner ses propriétés à partir de la fenêtre Propriétés. Ce nouveau contrôle est visible lors de l'exécution, mais vous pouvez aussi créer un contrôle à l'arrière-plan qui ne s'affiche pas dans la fenêtre feuille lors de l'exécution

(comme c'est le cas du contrôle Timer). Il faut pour cela positionner la propriété `InvisibleAtRunTime` à `True`. Un tel contrôle fonctionne à l'arrière-plan et n'apparaît pas sur la feuille utilisateur. (La propriété `Visible` détermine simplement si le contrôle peut être vu, alors que la propriété `InvisibleAtRunTime`, lorsqu'elle prend la valeur `True`, garantit que le contrôle ne peut jamais être affiché.)

Le nom du nouveau contrôle sera `TextSizeUL`.

L'importance des classes

Tous les objets Visual Basic, y compris les variables et les contrôles, sont membres d'une *classe*. Le groupement en classes vous offre une manière de regrouper les objets identiques. De plus, vous pouvez *sous-classer* un objet dans une classe existante pour créer un nouvel objet qui aura toutes les propriétés du reste de la classe et de nouvelles qui lui seront propres. Grâce au sous-classement, vous n'avez pas à réinventer la roue, car l'objet (dans ce cas le contrôle ActiveX) accepte automatiquement les propriétés, les méthodes et les événements supportés par la classe parente ; vous pouvez alors en ajouter de nouveaux.

Une classe est un regroupement (ou en fait une description) des propriétés, des méthodes et des événements supportés par un objet. Un contrôle n'est rien d'autre qu'une instance d'un objet d'une classe donnée. Une feuille ajoutée à un projet, par exemple, est simplement une instance que vous avez ajoutée à la classe `Form`. Le fait d'être un membre de la classe `Form` donne à la feuille ses propriétés, ses méthodes et ses événements, qui sont distincts des autres classes d'objets, par exemple la classe `CommandButton`.

Sous-classer signifie créer un nouvel objet à partir d'une classe d'objets existants. Le nouvel objet prend les propriétés, les méthodes et les événements de sa classe parente. Vous pouvez également y ajouter vos propres propriétés, méthodes et événements.

Les types de données intégrés, comme `String`, ne sont pas membres d'une classe Visual Basic, car ils ne supportent pas d'événements.

Supposons que vous vouliez créer un contrôle qui en imite en partie un autre. En sous-classant ce nouveau contrôle, vous lui adjoignez automatiquement toutes les propriétés, méthodes et événements supportés par le contrôle parent. Le sous-classement n'est cependant pas une obligation, car vous pouvez toujours créer un contrôle sans cette aide.

Visual Basic propose trois manières de créer de nouveaux contrôles ActiveX :

- **Les contrôles sous-classés simples**. Vous pouvez utiliser un contrôle ActiveX existant comme base de votre nouveau contrôle, qui est alors sous-classé à partir du contrôle d'origine (parent). Le nouveau contrôle reçoit toutes les fonctionnalités du contrôle parent existant et peut les étendre en ajoutant de nouvelles caractéristiques. Le sous-classement d'un contrôle est la manière la plus simple de créer un contrôle ActiveX. Vous devez modifier l'interface du contrôle sous-classé pour qu'il supporte les nouvelles caractéristiques que vous voulez que votre contrôle offre.
- **Les contrôles sous-classés agrégés**. Vous pouvez sous-classer votre nouveau contrôle à partir de plusieurs contrôles existants. En d'autres termes, si votre nouveau contrôle ressemble à une boîte de dialogue contenant des boutons de commande, des boîtes de texte, des étiquettes, vous pouvez utiliser les contrôles existants pour réduire le travail à effectuer sur le nouveau contrôle. Vous pouvez alors vous concentrer sur les fonctionnalités supplémentaires qu'il doit apporter.
- **Les contrôles dessinés par l'utilisateur**. Si votre contrôle n'a rien en commun avec des contrôles existants, vous pouvez le créer à partir de zéro en définissant toutes ses propriétés, tous ses événements et méthodes, puis le dessiner pour qu'il ressemble exactement à ce que vous souhaitez. Un contrôle créé par l'utilisateur demande un réel effort de création, car on n'emprunte aucune des fonctionnalités des contrôles existants.

Créer le contrôle ActiveX

Les sections qui suivent vous guident dans le processus de création du nouveau contrôle ActiveX TextSizeUL.

Préparer Visual Basic

Vous ne devez pas suivre la procédure standard de création des applications Visual Basic lorsque vous créez un contrôle ActiveX. Lorsque vous sélectionnez Nouveau, Projet pour afficher la boîte de dialogue Nouveau Projet, au lieu de cliquer sur l'icône EXE standard, sélectionnez l'icône Contrôle ActiveX. Visual Basic se prépare pour le nouveau contrôle et l'écran ressemble beaucoup à celui de la création d'une nouvelle application — mis à part que la fenêtre feuille semble s'afficher sans sa bordure habituelle (voir Figure 17.4). En réalité, la fenêtre ne montre pas une feuille, mais la toile de fond du contrôle ActiveX à créer. Visual Basic lui attribue le nom par défaut `UserControl1`. Naturellement, le mot User est trompeur, car c'est en tant que programmeur que vous créez ce contrôle pour vous assister dans votre développement. Une fois le contrôle conçu, cependant, l'utilisateur final pourra effectivement interagir avec lui.

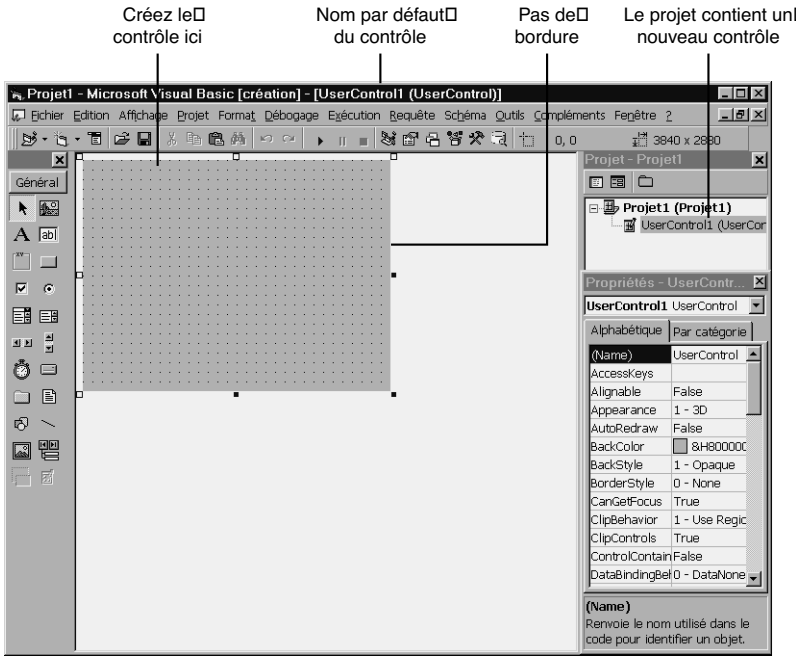


Figure 17.4
Vous travaillez dans un environnement familier lorsque vous créez un contrôle ActiveX.

Le terme *exécution* a un sens différent suivant que vous créez des contrôles ou des applications classiques. Alors que la conception d'un contrôle se produit quand vous créez ou modifiez le contrôle, il peut être exécuté dans les deux situations suivantes :

- Quand un programmeur l'insère au moment de la conception d'une application, le contrôle est un exécutable compilé qui réagit aux instructions d'installation du programmeur.
- Lorsque le programmeur compile et exécute l'application, le contrôle s'exécute aussi, mais il réagit alors à un utilisateur final.

Pour distinguer ces deux types d'exécution, les programmeurs parlent respectivement d'exécution à la conception et d'exécution à l'exécution. En général, le contexte du mode d'exécution est évident quand vous créez ou utilisez le contrôle ActiveX.

L'exécution à la conception est le terme qui s'applique à un contrôle que vous "exécutez" pour le tester au moment de sa création. L'exécution à l'exécution s'applique au contrôle qui s'exécute de concert avec une application.

Commencer la personnalisation

Vous pouvez ajouter votre nouveau contrôle ActiveX à la fenêtre Boîte à outils. Une des premières choses à faire est donc, une fois les objectifs du contrôle définis, de lui affecter une icône. La propriété `ToolboxBitmap` configure l'icône que vous voulez utiliser. Le rectangle gris qui ressemble un peu à une fenêtre feuille dans le coin supérieur gauche de la zone d'édition est en fait votre contrôle, mais il est encore vierge. Lorsqu'elle est sélectionnée, la fenêtre Propriétés affiche donc des valeurs pour le contrôle.

Faites défiler la fenêtre des propriétés pour trouver `ToolboxBitmap`. Lorsque vous double-cliquez sur la propriété, la boîte de dialogue Charger un bitmap s'ouvre pour vous permettre de trouver une icône sur le disque et l'utiliser comme icône du contrôle ActiveX. Dans notre cas, sélectionnez dans le répertoire `\Graphics\Bitmaps\Assorted \Plan`, qui se trouve dans le dossier `Common` de Visual Basic (si vous avez installé les fichiers graphiques supplémentaires lors de l'installation de Visual Basic).

Vous pouvez créer votre propre image bitmap avec Windows Paint (ou un programme de dessin du même genre). L'image doit avoir une taille de 15 pixels par 16 pour qu'elle s'affiche de la même manière que les autres icônes de la boîte à outils.

L'image Plan est une icône qui s'adaptera bien à cette nouvelle entrée de boîte à outils qu'est la nouvelle zone de texte. Le bitmap s'affichera dans le coin supérieur gauche de la feuille du contrôle lorsque vous renseignez la propriété `ToolboxBitmap`.

Les informations concernant votre contrôle s'afficheront dans les applications qui en feront usage, il est donc utile de le documenter. Sélectionnez Projet, Propriétés et modifiez le nom du projet de `Project1` en `TextSizeUL`. Ajoutez une description dans la zone de texte intitulée Description du projet pour que les applications qui utiliseront le contrôle puissent l'afficher. Tapez ce qui suit dans la description :

```
Un contrôle de zone de texte qui commande sa propre taille et
➥gère la conversion majuscule-minuscule.
```

Modifiez le nom de propriété du contrôle en `NewControl`. Enregistrez ensuite le contrôle et le projet lorsque la question est posée. Visual Basic enregistre le contrôle sous le nom `NewControl.ctl`, mais vous finirez par convertir le projet en un fichier ActiveX.

Sous-classer le contrôle

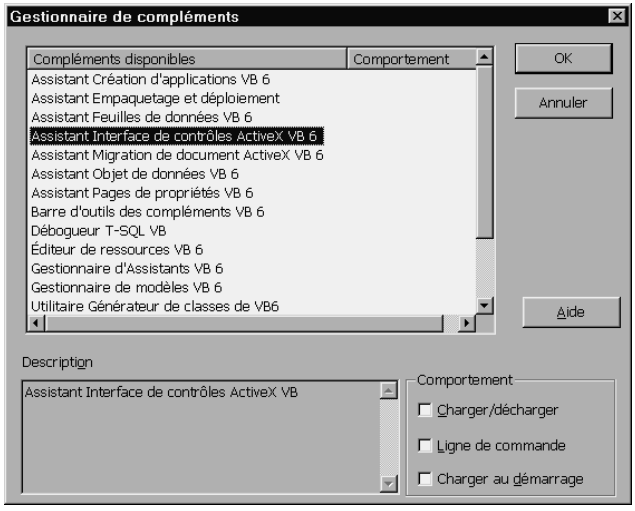
Vous pouvez désormais sous-classer le nouveau contrôle à partir du contrôle TextBox classique, pour qu'il en prenne les propriétés, les événements et les méthodes. Lors du sous-classement du contrôle, l'assistant Interface de contrôles ActiveX vous permettra d'étendre ses fonctionnalités.

L'assistant Interface de contrôles ActiveX est un assistant qui vous guide dans le processus du sous-classement d'un nouveau contrôle.

L'assistant Interface de contrôles ActiveX ne fait pas partie de l'environnement Visual Basic par défaut. Vous pouvez l'y ajouter par les étapes suivantes :

1. Sélectionnez le menu Compléments.
2. Sélectionnez le Gestionnaire de compléments pour afficher la boîte de dialogue correspondante, illustrée à la Figure 17.5.

Figure 17.5
Ajouter le complément Assistant Interface de contrôles ActiveX à votre environnement.



3. Double-cliquez sur l'élément "Assistant Interface de contrôles ActiveX". Le message "Chargé" qui apparaît à sa droite vous informe que l'assistant fait désormais partie de l'environnement.

Si vous créez souvent des contrôles ActiveX, vous pouvez cocher la case Charger au démarrage pour que l'assistant se charge toujours avec l'environnement Visual Basic.

4. Cliquez sur OK pour fermer la boîte de dialogue.

Si l'assistant est bien chargé, vous n'êtes pas encore prêt à le démarrer ; vous devez commencer par ajouter un contrôle TextBox à sous-classer dans votre contrôle ActiveX vierge. Vous pouvez cependant vérifier que l'assistant a bien été ajouté à

l'environnement Visual Basic en affichant le menu Compléments. La dernière option doit être Assistant Interface de contrôles ActiveX. Chaque fois que vous ajoutez des composants à l'environnement Visual Basic, ils s'affichent dans le menu Compléments.

Avant de démarrer l'assistant, vous devez ajouter un contrôle TextBox à votre projet. Placez-le sur l'arrière-plan et nommez-le `txtParent`. Cette zone de texte sera commandée par le nouveau contrôle ActiveX. Lorsqu'un programmeur utilisera votre contrôle ActiveX et le redimensionnera, ce sont les procédures événementielles qui modifieront la zone de texte interne du nouveau contrôle. Le contrôle TextBox fournit les fonctionnalités que vous devez sous-classer.

D'une certaine manière, la zone de texte agira comme un contrôle masqué (exactement comme un objet local dans un contrôle public), qui sera utilisé par d'autres projets. Votre contrôle utilisera ce contrôle intrinsèque, en modifiera le comportement et le présentera alors à d'autres applications.

Démarrer l'assistant

Vous êtes maintenant prêt à lancer l'assistant Interface de contrôles ActiveX à partir du menu Compléments. La première fenêtre qui s'affiche est un écran d'introduction que vous avez le choix de cacher lors d'exécutions ultérieures en cochant la case en bas de la fenêtre. Après avoir lu le contenu de cette fenêtre, cliquez sur Suivant pour afficher la fenêtre Sélection des membres d'interface, illustrée à la Figure 17.6.

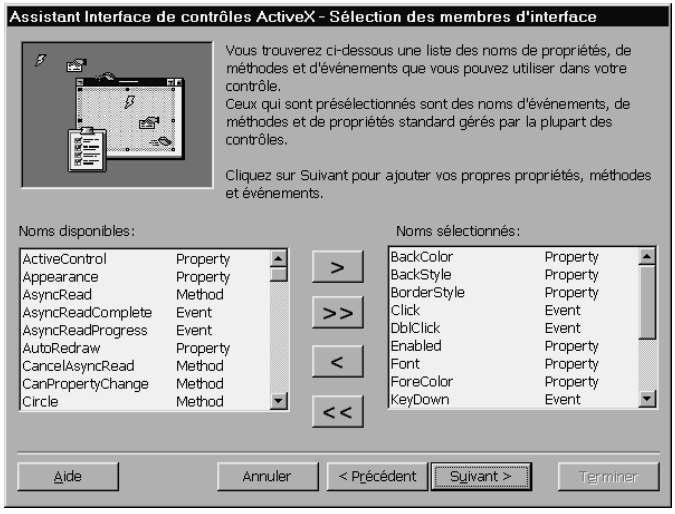


Figure 17.6
Vous pouvez sélectionner les éléments du contrôle parent que vous voulez inclure dans le nouveau contrôle ActiveX.

L'assistant affiche deux listes d'informations. Celle de gauche contient tout ou partie des propriétés, événements et méthodes que vous pouvez inclure dans le nouveau contrôle ActiveX. L'assistant génère la liste en analysant le contrôle en cours ; ici, le contrôle TextBox placé dans la zone d'édition.

La liste de droite contient plusieurs propriétés, méthodes et événements sélectionnés par l'assistant dans la liste complète de gauche. Vous pouvez sélectionner des éléments supplémentaires en cliquant sur un élément de la fenêtre de gauche, puis sur le bouton > pour l'envoyer dans la liste de droite des éléments sélectionnés. Vous pouvez aussi sélectionner des éléments dans la liste de droite et cliquer sur < pour les supprimer.

Le bouton >> envoie tous les éléments de la liste de gauche dans la liste de droite. Le bouton << efface tous les éléments sélectionnés pour vous permettre de repartir de zéro.

D'une manière surprenante, l'assistant n'envoie pas plusieurs propriétés nécessaires aux boîtes de texte sous-classées dans la liste de droite ; vous devez donc sélectionner et faire passer à droite les éléments suivants :

- Alignment
- Change
- FontBold
- FontItalic
- FontName
- FontSize
- FontStrikethru
- FontUnderline
- MultiLine
- PasswordChar
- ScrollBars
- Text
- ToolTipText

Vous devez pouvoir appliquer toutes ces propriétés à votre nouveau contrôle ActiveX de zone de texte, avec les nouvelles propriétés à définir. Une fois les éléments ajoutés, cliquez sur Suivant pour afficher la fenêtre Création des membres d'interface personnalisés, illustrée à la Figure 17.7.

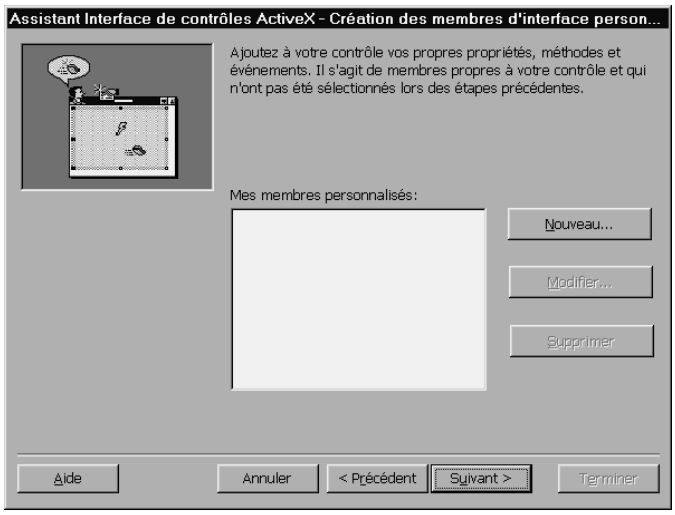


Figure 17.7
Vous pouvez maintenant ajouter vos propriétés, méthodes et événements propres.

La liste au centre de la fenêtre est vierge, car il vous reste à ajouter vos propres éléments à la configuration du contrôle. Vous ne pouvez pas taper directement l'élément, vous devez cliquer sur le bouton Nouveau pour décrire la nouvelle propriété (ou méthode ou événement) que vous souhaitez ajouter.

En cliquant sur Nouveau, vous affichez la boîte de dialogue illustrée à la Figure 17.8. Vous devez saisir le nom et le type (propriété, méthode ou événement) du nouvel élément en cliquant sur l'une des options présentées. Puis cliquez sur OK pour l'ajouter à la liste de l'assistant.

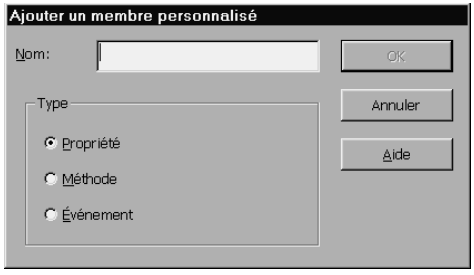


Figure 17.8
Description des nom et type du nouvel élément.

Pour le contrôle ActiveX que vous êtes en train de créer, entrez les deux éléments suivants :

- `AutoTSize` (propriété)
- `ULText` (propriété)

Cliquez sur Suivant pour passer à la fenêtre suivante, Définition de l'association. C'est là que vous associez, ou connectez, les propriétés, événements et méthodes aux équivalents de la zone de texte. En d'autres termes, la liste affichée dans la fenêtre Noms publics est celle qui est générée à partir des éléments sélectionnés précédemment et des nouveaux que vous avez ajoutés dans la fenêtre précédente.

Vous devez indiquer à l'assistant comment vous voulez que chaque propriété, méthode et événement se comporte. Dans le cas de la nouvelle zone de texte que vous créez, vous voulez que tous fonctionnent comme ils le font normalement pour une zone de texte. (Vous pourriez cependant modifier les associations : par exemple, un événement `MouseDown` pourrait être associé à un événement `MouseClick`, généré à la place de l'événement `MouseDown` normal qui se produit quand l'utilisateur clique sur le contrôle.)

Les deux seules propriétés que vous ne voulez pas associer au comportement normal d'une zone de texte sont les deux que vous avez ajoutées : `AutoTSize` et `ULText`. Sélectionnez donc tous les éléments de la liste hormis ces deux-là.

La liste supportant les sélections multiples, une manière rapide de tout sélectionner sauf vos deux propriétés consiste à cliquer sur le premier élément de la liste, puis à appuyer sur les touches Maj+Fin pour mettre tout en surbrillance. En maintenant la touche Ctrl enfoncée, cliquez enfin sur les propriétés `ULText` et `AutoTSize` pour les désélectionner.

Sélectionnez l'élément auquel vous voulez associer tous ces éléments. Ouvrez la liste déroulante Correspond à Contrôle et sélectionnez `txtParent`. Tous ces éléments publics, à l'exception des deux qui ne sont pas sélectionnés, sont appliqués au contrôle TextBox incorporé que vous avez placé sur la feuille. Vous voulez que la zone de texte interne se comporte normalement pour que le contrôle ActiveX puisse y accéder normalement. Les deux nouvelles propriétés ne s'appliqueront absolument pas à la zone de texte interne — c'est pourquoi elles n'y sont pas associées.

Cliquez sur Suivant pour afficher la fenêtre Définition des attributs illustrée à la Figure 17.9. C'est ici que vous faites correspondre les nouvelles propriétés au nouveau contrôle.

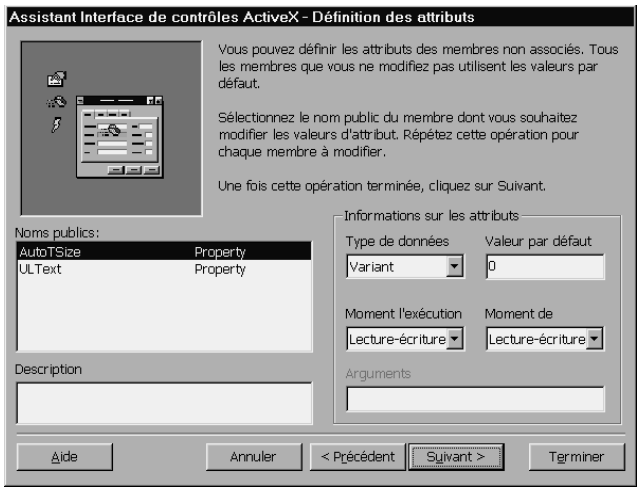


Figure 17.9
Vous pouvez maintenant faire correspondre les nouvelles propriétés à votre contrôle.

Si vous avez ajouté des méthodes et des événements au nouveau contrôle, ils apparaîtront également dans la fenêtre Définition des attributs. Vous avez déjà fait correspondre les propriétés, méthodes et événements connus au contrôle interne, vous devez maintenant vous occuper des nouveaux. Suivez ces étapes pour préparer les nouvelles propriétés :

1. Pour la propriété `AutoTSize` sélectionnée, changez la valeur par défaut de `0` à `1` (les autres champs sont déjà corrects). Si vous vous en souvenez, la propriété `AutoTSize` devra prendre une valeur parmi quatre énumérées, qui vont de `1` à `4`. La valeur `1` sera la valeur par défaut (c'est-à-dire la valeur prise lorsqu'on place pour la première fois le contrôle ActiveX dans une feuille). Il faut donc remplacer le `0` par `1`.

2. Tapez la description suivante pour la propriété `AutoTSize` :

```
Détermine le pourcentage (25%, 50% ou 75%) de la taille de police du
➥texte en fonction de la hauteur.
```

3. Sélectionnez la propriété `ULText`. Tous les champs sont corrects, mais il faut ajouter cette description :

```
Met le texte en majuscules, minuscules ou ne le change pas.
```

4. Cliquez sur Suivant pour achever la tâche de l'assistant. Il peut désormais construire toutes les informations nécessaires à la génération du contrôle. Vous pouvez cliquer sur Terminer pour le générer. Si vous laissez cochée l'option Afficher le récapitulatif, l'assistant affiche un résumé de ce qui reste à faire.

A l'évidence, le contrôle ActiveX est incomplet. Vous n'avez pas défini le comportement des propriétés `AutoTSize` et `ULText`. L'assistant configure les paramètres du contrôle, mais vous devez toujours ajouter du code et compléter manuellement le contrôle ActiveX pour le rendre entièrement fonctionnel.

Compléter le corps du contrôle ActiveX

L'assistant ne peut pas ajouter le code qui active les propriétés `AutoTSize` et `ULText`, car il n'a aucun moyen de savoir ce que vous voulez en faire. Il a cependant pu associer les propriétés, méthodes et événements existants au contrôle parent, car ces éléments sont déjà définis.

Pour achever le contrôle ActiveX, vous devez faire un peu de programmation, ce qui, naturellement, se fait dans la fenêtre Code. Sélectionnez Affichage, Code pour voir le contenu de la fenêtre. En parcourant le code, vous remarquerez que la majeure partie est dédiée à l'association des propriétés, méthodes et événements, à ceux de la zone de texte `txtParent` sous-jacente. En d'autres termes, quand un programmeur utilisant ce nouveau contrôle configure, en exécution de conception (c'est-à-dire que le code s'exécute lors de la conception d'une autre application), la propriété `BackColor`, le code configure en fait la propriété `BackColor` de la zone de texte sous-jacente. Ce code est parfois complexe, attendez-vous donc à ne pas tout en comprendre.

Observez, dans le code, toutes les procédures événementielles `Let` et `Get`. Comme vous devez vous en souvenir, ces deux qualifiants particuliers de fonctions sont utilisés pour configurer et renvoyer des valeurs pour les propriétés que vous créez. L'assistant a en effet créé des propriétés pour le contrôle TextBox parent lorsque vous les avez associées au nouveau contrôle.

Ne faites jamais de modifications dans les parties du code préfacées par la remarque suivante :

```
'ATTENTION! NE SUPPRIMEZ PAS OU NE MODIFIEZ PAS LES LIGNES
➥COMMENTEES SUIVANTES!
```

Complétez toutes les parties qui contiennent des commentaires commençant par les mots : A COMPLETER. Vous verrez de telles parties si vous créez de nouveaux contrôles ActiveX basés sur un des contrôles de liste. L'assistant ne peut pas gérer les propriétés des listes, vous devez donc ajouter le code nécessaire au traitement des listes si vous utilisez un jour un contrôle de liste comme parent.

Les instructions qui suivent ces commentaires sont cruciales pour le fonctionnement du nouveau contrôle et doivent êtres laissées telles quelles.

Le Listing 17.2 montre les premières lignes du code. Elles sont dédiées aux nouvelles propriétés que vous avez ajoutées au projet.

Listing 17.2 : L'assistant a initialisé les valeurs par défaut des nouvelles propriétés

```
1: 'Valeurs de propriétés par défaut:
2: Const m_def_AutoTSize = 1
3: Const m_def_ULText = 0
4: 'Variables de propriétés:
5: Dim m_AutoTSize As Variant
6: Dim m_ULText As Variant
```

Le mot clé `Const` déclare des constantes nommées. Donc, à la ligne 2, `m_def_AutoTSize` n'est pas une variable, mais une constante nommée. Visual Basic comporte plusieurs constantes nommées telles que `vbWhite` et `vbInformation` ; vous pouvez aussi déclarer les vôtres, qui peuvent être locales ou globales. (Les constantes nommées sont souvent globales, car, de par leur nature même, elles ne risquent pas d'être modifiées par inadvertance par une procédure qui ne devrait pas y avoir accès.)

Les lignes 2 et 3 déclarent des constantes nommées pour les valeurs par défaut des deux nouvelles propriétés. Elles sont configurées dans la fenêtre Définir les attributs de l'assistant. Elles apparaîtront dans la fenêtre Propriétés correspondant aux deux propriétés quand un programmeur placera le contrôle ActiveX dans une feuille. Chaque fois que le reste du code fait référence à une de ces constantes nommées, elle est remplacée par sa valeur créée aux lignes 2 et 3 du Listing 17.2.

Les lignes 5 et 6 déclarent des variables Variant qui représentent la valeur en cours des propriétés, définie quand un programmeur configure une de ces valeurs en exécution de conception (ou quand l'application finale paramètre par programme ces valeurs).

Vous devez définir les valeurs énumérées des deux propriétés qui s'afficheront dans la fenêtre Propriétés. Une liste énumérée se décrit dans un pavé de code d'énumération qui commence par l'instruction `Enum`. Juste après la partie générale de la fenêtre de code, tapez le code du Listing 17.3.

Listing 17.3 : Vous devez définir les valeurs énumérées qui s'afficheront dans la fenêtre Propriétés

```
1:  Public Enum AutoTSizeEnum
2:    NA = 1
3:    Small = 2
4:    Medium = 3
5:    Large = 4
6:  End Enum
```

```
 7:  Public Enum ULTextEnum
 8:    AsIs = 0
 9:    Uppercase = 1
10:    Lowercase = 2
11: End Enum
```

Déclarez toutes les valeurs énumérées publiques pour qu'elles soient toujours accessibles. L'instruction `Enum` débute une définition de valeurs énumérées. Ce sont des valeurs qui s'afficheront dans une liste déroulante de la fenêtre Propriétés pour les deux propriétés. Les valeurs `AutoTSize` se présenteront sous la forme courante suivante :

```
1 - NA
2 - Small
3 - Medium
4 - Large
```

Pour paramétrer une valeur initiale, le programmeur peut sélectionner une de ces valeurs dans la fenêtre Propriétés lorsqu'il travaille avec le contrôle ActiveX. En outre, le code inclus dans l'application qui utilise ce contrôle ActiveX peut assigner `1`, `2`, `3` ou `4` à la propriété pour paramétrer le contrôle. L'instruction d'assignation peut également assigner les valeurs énumérées, comme on le fait ci-dessous :

```
NewControl.AutoTSize = Medium  ' Assigne 3
```

De la même manière, le type énuméré `ULTextEnum` définit les valeurs énumérées de la propriété `ULText`.

Le code de dimensionnement du nouveau contrôle est très simple, car il doit prendre la taille de sa zone de texte parente. Souvent, un nouveau contrôle ActiveX doit prendre une taille différente du ou des contrôles qu'il sous-classe, en particulier avec les contrôles ActiveX sous-classés agrégés. Cependant, lorsqu'il y a correspondance univoque entre la taille du parent et celle du nouveau contrôle, vous pouvez ajouter une procédure événementielle `UserControl_Resize()` en saisissant dans la fenêtre de code la procédure illustrée dans le Listing 17.4.

Listing 17.4 : Le nouveau contrôle ActiveX sera à la même place et de la même taille que le contrôle TextBox interne

```
1:  Private Sub UserControl_Resize()
2:    ' Définit la hauteur et l'échelle à celle du contrôle sous-jacent
3:    ' Etend le contrôle aux bonnes largeur et hauteur
4:    If UserControl.Height <> txtParent.Height Then
5:       txtParent.Height = UserControl.Height
6:    End If
7:    txtParent.Move 0, 0, UserControl.ScaleWidth
8:  End Sub
```

La ligne 4 permet de s'assurer que, lorsque le programmeur qui utilise le nouveau contrôle modifie sa taille, le contrôle TextBox incorporé est également redimensionné, car il fonctionne comme un réceptacle du nouveau contrôle. La ligne 7 utilise la méthode `Move` pour déplacer la zone de texte parente vers le coin supérieur gauche du nouveau contrôle (0, 0), puis paramètre la même échelle pour les deux propriétés. Cela garantit que la zone de texte interne se déplace chaque fois que le programmeur déplace le nouveau contrôle. La méthode `Move` gère à la fois le déplacement et le dimensionnement et permet de conserver les deux contrôles à l'identique. La zone de texte agit donc comme un calque sur le contrôle ActiveX et reste toujours exactement au-dessus pour recevoir et afficher le texte.

Maintenant que vous êtes débarrassés du problème de la taille, votre tâche principale consiste à configurer l'affichage des valeurs énumérées, à écrire le code qui gère la sélection de la propriété `AutoTSize` et la conversion du contrôle ActiveX en minuscules ou majuscules, qui dépend de la propriété `ULText`. L'assistant a créé un réceptacle pour ces propriétés, mais vous devez compléter les détails.

Vous êtes maintenant prêt à configurer les procédures `Let` et `Get` pour les nouvelles valeurs de propriétés. Lorsque l'utilisateur (c'est-à-dire le programmeur qui utilise ce contrôle ActiveX dans une application) paramètre une valeur de propriété, la procédure `Let` s'exécute. Lorsque l'utilisateur accède à une valeur de propriété, la procédure `Get` s'exécute.

La méthode la plus simple des deux est `Get`. L'assistant a créé des procédures échantillons de la fonction `Get` pour les deux propriétés, mais vous devrez les modifier. Au lieu de renvoyer un type de données Variant, il faudra utiliser le type énuméré, comme le montre le Listing 17.5.

Listing 17.5 : Les procédures Get des nouvelles propriétés doivent renvoyer les valeurs énumérées correspondantes

```
1:  Public Property Get AutoTSize() As AutoTSizeEnum
2:      AutoTSize = m_AutoTSize
3:  End Property
4:
5:  Public Property Get ULText() As ULTextEnum
6:     ULText = m_ULText
7:  End Property
```

Ce listing ne fait qu'assigner la valeur en cours du membre à la propriété d'état. La conversion de Variant en énumération pour la valeur de retour est la seule modification nécessaire.

Vous devez aussi modifier les deux types de données des procédures `Let`. Elles demandent du code supplémentaire. Lorsqu'une valeur est assignée à une des deux nouvelles propriétés, plusieurs choses doivent se passer, par exemple le dimensionnement du texte ou la conversion en majuscules ou en minuscules.

Il vous faut donc compléter les deux procédures `Let` des propriétés correspondantes, car l'assistant n'a créé qu'un noyau de code. Le code résultant est montré dans le Listing 17.6.

Listing 17.6 : Vous devez compléter les procédures Let des deux propriétés

```
 1:  Public Property Let AutoTSize(ByVal New_AutoTSize As AutoTSizeEnum)
 2:      m_AutoTSize = New_AutoTSize
 3:      ' Tester l'état de la propriété et en modifier la taille
 4:      ' en fonction de sa valeur
 5:      '
 6:      Select Case New_AutoTSize
 7:         Case 1: ' Pas de modification nécessaire
 8:         Case 2: Font.Size = 72 * 0.25 * (Height / 1440)
 9:         Case 3: Font.Size = 72 * 0.5 * (Height / 1440)
10:        Case 4: Font.Size = 72 * 0.75 * (Height / 1440)
11:     End Select
12:     PropertyChanged "AutoTSize"
13: End Property
14:
15: Public Property Let ULText(ByVal New_ULText As ULTextEnum)
16:     m_ULText = New_ULText
17:     ' Tester l'état du contrôle
18:     ' et modifier en fonction la zone de texte
19:     ' (ignorer ULText à 0 qui signifie tel quel)
20:     If New_ULText = 1 Then
21:       Text = UCase(txtParent.Text)
22:     ElseIf New_ULText = 2 Then
23:       Text = LCase(txtParent.Text)
24:     End If
25:     PropertyChanged "ULText"
26: End Property
```

Assurez-vous que vous avez bien modifié les types de données transmis des procédures `Get AutoTSize()` et `Get ULText()` pour qu'elles reçoivent des données énumérées au lieu des types de données Variant qu'elles reçoivent par défaut.

Les lignes 8, 9 et 10 ajustent la taille de police du contrôle TextBox interne à un facteur de sa propriété `Height`. Le programmeur travaille directement sur le nouveau contrôle ActiveX, mais ce dernier n'est en fait qu'un intermédiaire vers la zone de texte interne qui s'affiche dans la feuille. Les fonctions internes `UCase()` et `LCase()` des lignes 21 et 23 convertissent le texte en majuscules ou minuscules, suivant la valeur donnée à la propriété. Si le programmeur qui utilise le contrôle ActiveX assigne la propriété `ULText` à la conception ou à l'exécution, cette procédure s'exécute.

La conception et la création du contrôle ActiveX est achevée. Vous devez maintenant le préparer à être inséré dans une autre application et le tester pour vous assurer qu'il fonctionne comme vous le voulez.

Implémenter le contrôle ActiveX

Le nouveau contrôle ActiveX peut être inséré dans une application et placé sur la feuille, et il reçoit également tous les avantages des contrôles intrinsèques. Sa fenêtre Propriétés fonctionnera comme pour les autres contrôles. De même, quand un programmeur qui utilise le contrôle ActiveX dans une fenêtre de code tape une instruction d'assignation d'une valeur de propriété, même la fenêtre Info Express s'affiche pour permettre de la sélectionner. La boîte à outils qui contient le contrôle ActiveX affiche aussi automatiquement une info-bulle qui le décrit. Vous serez fiers de votre contrôle ActiveX lorsque que vous l'utiliserez, car il se comportera comme s'il avait été fourni avec Visual Basic par Microsoft !

Quand vous compilez le contrôle ActiveX, Visual Basic le transforme en un fichier ActiveX que vous pouvez insérer dans un projet comme n'importe quel contrôle ActiveX. Si vous ne créez pas le fichier OCX, vous ne pourrez pas utiliser le contrôle dans une application.

Le contrôle doit être enregistré avant compilation. Choisissez Fichier, Enregistrer le projet pour enregistrer à la fois le contrôle et le projet. Un contrôle ActiveX ne peut pas être exécuté en utilisant la touche F5, car il doit être compilé avant de pouvoir s'exécuter. Le terme *exécuter* signifie en fait ici *fonctionner comme les autres* quand un programmeur utilise le contrôle dans une application.

Pour compiler le contrôle ActiveX, sélectionnez Fichier, Créer. Visual Basic affiche la boîte de dialogue Créer le projet illustrée à la Figure 17.10. Vous pouvez sélectionner l'emplacement du contrôle ActiveX compilé. Vous pouvez le placer dans votre dossier `\Windows\System` ou dans un dossier de travail Visual Basic que vous avez créé. (C'est le dossier dans lequel vous recherchez quand vous voulez charger le contrôle Visual Basic dans la boîte à outils d'une autre application Visual Basic à partir de la boîte de dialogue Propriétés.) Si le compilateur signale des erreurs, Visual Basic ne créera pas le contrôle et mettra en surbrillance les lignes de code en erreur. Dès que les bogues sont éliminés, le compilateur vous renvoie dans l'environnement de développement.

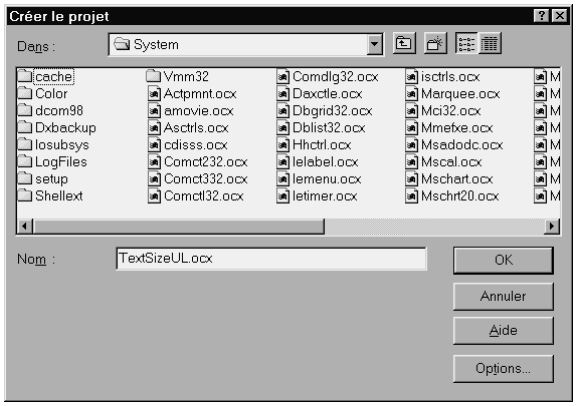


Figure 17.10
Entrez le nom de fichier du contrôle ActiveX ; Visual Basic enregistre le contrôle avec une extension .OCX.

Visual Basic supporte deux méthodes de test du contrôle :

- Ouvrir un nouveau projet et y tester le contrôle. L'environnement de développement n'est disponible que pour le test des contrôles ActiveX.
- Ouvrir un nouveau projet EXE Standard et y déposer le contrôle.

Pour tester le nouveau contrôle, il faut l'utiliser en situation. Sélectionnez Fichier, Nouveau Projet et créez un nouveau fichier EXE Standard. Appuyez sur Ctrl-T pour ouvrir la boîte de dialogue Composants. Comme le montre la Figure 17.11, le contrôle `TextSizeUL` apparaît en haut de la boîte de dialogue. La description que vous avez utilisée pour le contrôle s'affiche pour faciliter la sélection.

Sélectionnez le contrôle ActiveX et fermez la boîte de dialogue. L'image bitmap que vous avez sélectionnée à la création du contrôle s'affiche dans la boîte à outils. Pour utiliser le programme dans une application, suivez ces étapes :

1. Changez le nom de feuille en `frmActiveX` et son titre en `Test du contrôle ActiveX`. Etendez les propriétés `Width` et `Height` à 7575 et 5775, respectivement.
2. Pointez le contrôle `TextSizeUL` et lisez l'info-bulle créée par l'assistant de contrôle ActiveX. `NewControl` n'est pas très parlant, mais c'est le nom sous lequel vous avez enregistré le projet. Dans cette session, nous nous intéressons plus au fonctionnement qu'au nom du contrôle.
3. Double-cliquez sur le nouveau contrôle pour l'ajouter à la feuille. (Vous pouvez aussi dessiner la taille du contrôle à la souris.) Le contrôle ressemble à une zone de texte tout ce qu'il y a de plus normal mis à part ses deux propriétés supplémentaires. Dimensionnez le contrôle `TextSizeUL` à environ 4 815 twips (propriété `Width`) par 1 215 twips (propriété `Height`). Mettez la propriété `Font.Size` à `18` et `Font.Bold` à `True`.

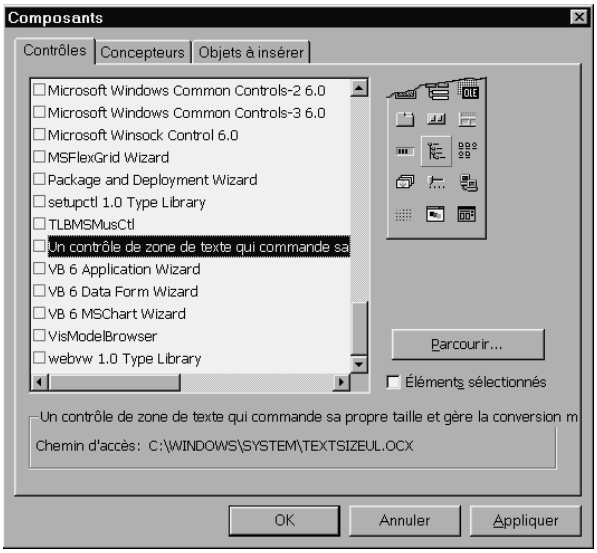


Figure 17.11
Le texte de description du nouveau contrôle ActiveX s'affiche dans la boîte de dialogue Composants.

4. Cliquez sur le bouton de la propriété `ULText` pour ouvrir la liste déroulante. Vous verrez trois valeurs énumérées, `AsIs`, `Uppercase` et `Lowercase`, telles qu'elles ont été programmées. Pour l'instant, laissez la valeur par défaut.
5. Cliquez sur la propriété `AutoTSize` pour voir ses valeurs énumérées. Laissez encore la valeur par défaut.
6. Changez la propriété `Name` en `MyFirstCtl` et effacez la propriété `Text`.
7. Ajoutez cinq boutons de commande à la feuille en utilisant les valeurs de propriétés du Tableau 17.1.

Tableau 17.1 : Définissez ces contrôles et ces propriétés pour les boutons de commande de la feuille

Contrôle	*Propriété*
Command1 `Name`	cmdSmall
Command1 `Caption`	Texte &Small
Command1 `Left`	1320
Command1 `Top`	2640
Command2 `Name`	cmdMedium

Tableau 17.1 : Définissez ces contrôles et ces propriétés pour les boutons de commande de la feuille *(suite)*

Contrôle	*Propriété*
Command2 `Caption`	Texte &Medium
Command2 `Left`	3120
Command2 `Top`	2640
Command3 `Name`	cmdLarge
Command3 `Caption`	Texte &Large
Command3 `Left`	4920
Command3 `Top`	2640
Command4 `Name`	cmdUpper
Command4 `Caption`	Maj&uscules
Command4 `Left`	2160
Command4 `Top`	3600
Command5 `Name`	cmdLower
Command5 `Caption`	Minuscu&les
Command5 `Left`	3960
Command5 `Top`	3600

8. Ajoutez les procédures événementielles du Listing 17.7

Listing 17.7 : Ces procédures événementielles permettront de tester le nouveau contrôle ActiveX

```
 1:  Private Sub cmdSmall_Click()
 2:    ' Test de la conversion Small
 3:    MyFirstCtl.AutoTSize = Small
 4:  End Sub
 5:
 6:  Private Sub cmdMedium_Click()
 7:    ' Test de la conversion Medium
 8:    MyFirstCtl.AutoTSize = Medium
 9:  End Sub
10:
11: Private Sub cmdLarge_Click()
```

Listing 17.7 : Ces procédures événementielles permettront de tester le nouveau contrôle ActiveX *(suite)*

```
12:   ' Test de la conversion Large
13:   MyFirstCtl.AutoTSize = Large
14: End Sub
15:
16: Private Sub cmdUpper_Click()
17:   ' Test de la conversion en majuscules
18:   MyFirstCtl.ULText = Uppercase
19: End Sub
20:
21: Private Sub cmdLower_Click()
22:   ' Test de la conversion en minuscules
23:   MyFirstCtl.ULText = Lowercase
24: End Sub
```

En tapant ces lignes, remarquez que Visual Basic vous aide à localiser les valeurs de propriétés du nouveau contrôle ActiveX lorsque vous tapez le signe égale, à l'aide d'une liste déroulante d'options ((voir Figure 17.12). Les choix proposés sont les seules options acceptables par Visual Basic. Cette liste de sélection a donc quelque chose d'incroyable, étant donné que vous n'avez absolument rien fait pour générer cette fonction.

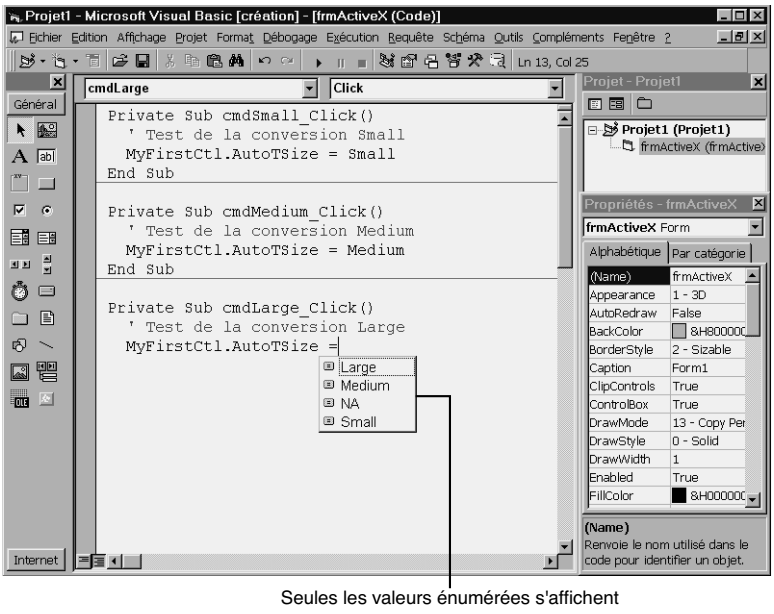


Figure 17.12
Votre contrôle ActiveX supporte la liste déroulante d'aide Info Express.

Compilez et exécutez l'application. Tapez une valeur dans la zone de texte du contrôle ActiveX, en combinant majuscules et minuscules. Cliquez sur les trois boutons de dimensionnement pour voir la taille du texte se modifier. Ces boutons ne sont pas associés à la propriété `Font.Size`, mais à la nouvelle propriété créée pour le contrôle. Cliquez aussi sur les boutons de commande de conversion de la casse pour voir les modifications.

Une fois que vous avez converti le texte du contrôle ActiveX en majuscules ou en minuscules, la casse d'origine est perdue.

Figure 17.13
Les propriétés du contrôle ActiveX effectuent désormais les conversions.

La Figure 17.13 montre l'application en exécution. Ces conversions ne demandaient pas la création d'un contrôle ; cependant, le nouveau contrôle supporte des propriétés intégrées qui peuvent être configurées à tout moment pour commander la taille du texte en fonction de la propriété `Height`, et également la casse du texte.

En résumé

La leçon d'aujourd'hui vous a expliqué comment travailler sur les objets particuliers que sont les contrôles ActiveX. Non seulement ils ajoutent des nouveaux contrôles à votre fenêtre Boîte à outils, mais ils peuvent être utilisés dans d'autres types d'applications Windows comme Visual C++ et les navigateurs Internet. C'est pourquoi les contrôles ActiveX sont disponibles à partir de nombreuses sources différentes. Une fois que vous avez appris à utiliser un contrôle ActiveX, vous pouvez le réutiliser dans d'autres applications.

Vous pouvez créer vos propres contrôles ActiveX en tirant avantage de l'assistant Interface de contrôles ActiveX. Vous pouvez sous-classer un contrôle ActiveX à partir d'un contrôle existant (même si c'est un autre contrôle ActiveX écrit par quelqu'un d'autre). Finalement, vous pourrez créer une bibliothèque de contrôles qui vous aideront dans la création de nouveaux contrôles et applications.

Questions-réponses

Q Pourquoi créer de nouveaux onglets de groupe dans la fenêtre Boîte à outils ?

R Les onglets ne sont là que pour organiser les contrôles. En les regroupant, vous retrouverez plus facilement le contrôle dont vous avez besoin pour un but particulier. Par exemple, si vous créez une application Visual Basic qui se rapporte aux bases de données, vous pouvez regrouper l'ensemble des contrôles qui ont trait à ce sujet dans leur propre onglet pour y accéder plus simplement. Vous n'avez plus à chercher dans tous les contrôles de l'onglet Général pour retrouver celui dont vous avez besoin. Même s'ils sont groupés dans les onglets, tous les contrôles de la fenêtre Boîte à outils sont toujours disponibles.

Q Puis-je me débarrasser des onglets de groupes que je crée ?

Oui. Il suffit de cliquer du bouton droit sur le nom de groupe pour afficher un menu contextuel contenant l'option Supprimer un onglet. Vous pouvez aussi renommer les onglets à partir de ce menu.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités et des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se trouvent à l'Annexe A.

Quiz

1. Que signifie *automatisation* ?
2. Que se passe-t-il si votre application utilise `CreateObject()` pour un document Word, alors que Word est déjà en exécution ?
3. Pourquoi ne peut-on pas assigner directement des applications à des variables objets ?
4. Quel est le but de l'objet système `Err.Number` ?
5. Quelles sont les trois manières de créer des contrôles ActiveX ?
6. Quelle est la méthode de création des contrôles ActiveX la plus simple à utiliser ?
7. Vrai ou Faux. Quand vous sous-classez un contrôle, le nouveau contrôle ActiveX emprunte les propriétés, les méthodes et les événements du parent.
8. A quoi servent les blocs d'énumérations ?
9. Quelle extension utilise Visual Basic pour les contrôles ActiveX compilés ?
10. Quelles sont les deux procédures obligatoires dans les propriétés des contrôles ActiveX ?

Exercices

1. Utilisez la boîte de dialogue Composants pour rechercher des contrôles ActiveX sur votre disque. Vous en trouverez certainement quelques-uns en dehors du dossier Visual Basic. Par exemple, si vous êtes membres du service en ligne Microsoft Network, vous trouverez plusieurs contrôles ActiveX dans le dossier Microsoft Network.
2. Modifiez le contrôle ActiveX et l'application créés à la fin de cette leçon. Modifiez la valeur énumérée `AsIs` en `AsEntered`. Modifiez le contrôle ActiveX pour que, quand un programme change le texte de `AsEntered` en `Uppercase` ou `LowerCase`, le contrôle mémorise le texte tel qu'il se présentait avant conversion. Récrivez l'application finale de cette leçon et ajoutez un sixième bouton de commande qui indique Comme saisi. Lorsque l'utilisateur clique dessus, le texte doit revenir à sa forme initiale.

PB8

Ces éléments qui enjolivent les applications

Ce projet décrit comment créer une application qui comporte les éléments suivants :

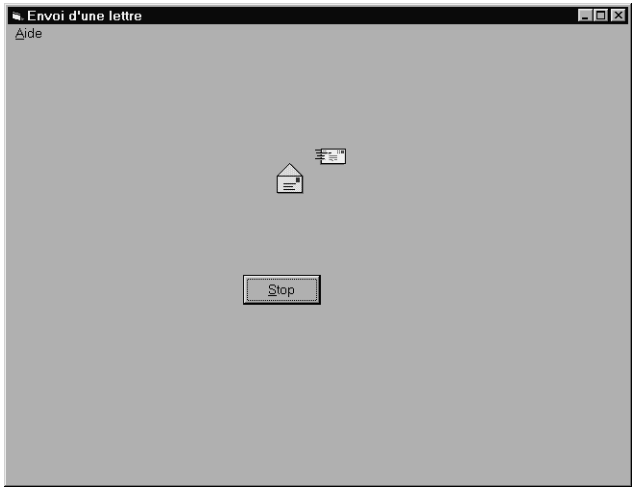
- une boîte A propos de affichée par l'option de menu Aide, A propos de ;
- un fichier son joué automatiquement lorsque la boîte A propos de s'affiche ;
- une zone d'image animée illustrant une animation simple ;
- un timer qui contrôle l'animation ;
- un groupe de contrôles images.

Ce projet ne comporte aucune fantaisie. Malgré la zone d'image animée, vous verrez que l'animation est assez triste. Cela ne pose cependant pas de problème, car le but est de vous faire comprendre les principes du déplacement des objets dans les zones d'images et vous aurez une expérience pratique de l'utilisation des boîtes A propos de et de l'ajout d'un fichier Wav à vos applications.

But de l'application

La Figure PB8.1 montre la feuille telle qu'elle se présentera durant l'animation. (A la première exécution de l'application, l'enveloppe est close et la lettre ne s'affiche pas). La feuille est simple et assez grande pour permettre à la lettre de s'échapper de l'enveloppe ouverte quand l'utilisateur clique sur le bouton Animer.

Figure PB8.1
L'application montre simplement une lettre qui s'échappe de l'enveloppe.



Une fois l'animation démarrée, la lettre continue à sortir de l'enveloppe, et le titre du bouton devient Stop.

La Figure PB8.2 montre la boîte A propos de qui s'affichera quand l'utilisateur sélectionne Aide, A propos de.

Création de la feuille principale

Le Tableau PB8.1 contient les contrôles et propriétés dont vous avez besoin pour créer la feuille principale qui s'affiche lorsque l'utilisateur démarre l'application. Vous devez avoir installé le dossier Graphics avec Visual Basic ; dans le cas contraire, vous devrez insérer le CD-ROM Visual Basic et pointer sur le dossier ou réinstaller Visual Basic avec les graphiques. De plus, vous devez appuyer sur Ctrl-T pour ajouter le contrôle multimédia à la boîte à outils avant de pouvoir l'insérer dans la feuille.

Figure PB8.2
Un fichier Wav est joué quand l'utilisateur affiche cette boîte A propos de.

De nombreux contrôles de ce projet, tel le contrôle Timer et les zones d'images, placés lors de la conception, se positionnent sur les bords extérieurs de la feuille, pour qu'ils ne gênent pas l'ajout des autres contrôles. Le contrôle Timer étant invisible à l'utilisateur, il peut être placé n'importe où. Les zones d'images se déplaçant dans l'animation, leur position initiale importe peu.

Ce projet contient trois contrôles de zones d'images qui font partie d'un unique groupe de contrôles. Vous pourriez créer trois contrôles PictureBox séparés, mais les mettre en groupe est un bon exercice pour les projets qui demandent de nombreux contrôles identiques en apparence et dans leur but. Pour créer le groupe, créez le premier contrôle, `picAni2` *(le contrôle* `picAni1` *est un contrôle indépendant). Une fois* `picAni2` *placé et ses propriétés assignées, copiez-le dans le Presse-papiers par la commande Edition, Copier. Sélectionnez Edition, Coller et répondez Oui quand la boîte de dialogue vous demande pour créer un groupe de contrôles. Visual Basic transforme le contrôle d'origine* `picAni2` *en* `picAni2(0)`*, premier élément du groupe. Copiez une nouvelle fois pour avoir un troisième élément dans le groupe* `picAni2`*.*

Pour vous donner une meilleure idée de l'apparence de la feuille, la Figure PB8.3 montre à quoi elle devrait ressembler après le placement des contrôles du Tableau PB8.1. A l'exécution, certains contrôles seront invisibles et donneront l'illusion d'une lettre qui jaillit de l'enveloppe.

Le son, Chimes.wav, qui est joué lorsque l'utilisateur sélectionne Aide, A propos de, est un fichier de son standard fourni avec Windows.

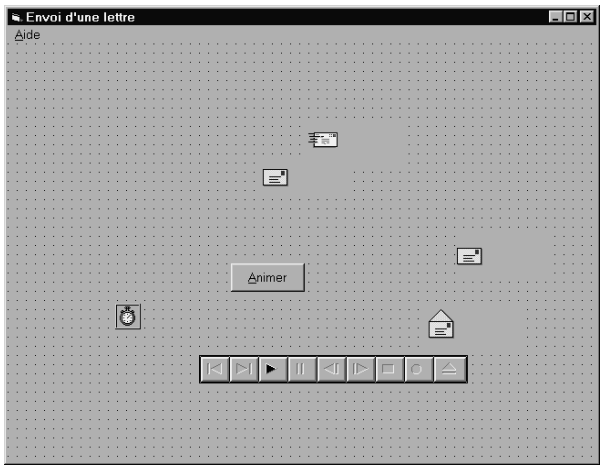


Figure PB8.3
Votre feuille aura cette apparence une fois les contrôles placés.

Tableau PB8.1 : Configurez ces contrôles et ces propriétés dans la feuille

Nom de propriété du contrôle	*Valeur de propriété*
Feuille : `Name`	frmEnvelope
Feuille : `Caption`	Envoyer une lettre
Feuille : `Height`	5790
Feuille : `Width`	7845
Option de menu #1 : `Name`	mnuHelp
Option de menu #1 : `Caption`	&Aide
Option de menu #2 : `Name`	mnuHelpAbout
Option de menu #2 : `Caption`	A &propos de…
Bouton de commande : `Name`	cmdAni
Bouton de commande : `Caption`	&Animate
Bouton de commande : `Left`	2940
Bouton de commande : `Top`	2880
Timer : `Name`	tmrAni

Tableau PB8.1 : Configurez ces contrôles et ces propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
Timer : `Enabled`	False
Timer : `Interval`	300
Timer : `Left`	1410
Timer : `Top`	3405
PictureBox #1 : `Name`	picAni1
PictureBox #1 : `Height`	495
PictureBox #1 : `Left`	3330
PictureBox #1 : `Picture`	Common\Graphics\Icons\Mail\Mail01a
PictureBox #1 : `Top`	1485
PictureBox #1 : `Width`	1215
PictureBox #2 : `Name`	picAni2(0)
PictureBox #2 : `Height`	495
PictureBox #2 : `Left`	5895
PictureBox #2 : `Picture`	Common\Graphics\Icons\Mail\Mail01a
PictureBox #2 : `Top`	2520
PictureBox #2 : `Width`	1215
PictureBox #3 : `Name`	picAni2(1)
PictureBox #3 : `Height`	495
PictureBox #3 : `Left`	5520
PictureBox #3 : `Picture`	Common\Graphics\Icons\Mail\Mail0lb
PictureBox #3 : `Top`	3240
PictureBox #3 : `Visible`	False
PictureBox #3 : `Width`	1215

Tableau PB8.1 : Configurez ces contrôles et ces propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
PictureBox #4 : `Name`	picAni2(2)
PictureBox #4 : `Height`	495
PictureBox #4 : `Left`	3960
PictureBox #4 : `Picture`	Common\Graphics\Icons\Mail\Mail03
PictureBox #4 : `Top`	1080
PictureBox #4 : `Visible`	False
PictureBox #4 : `Width`	1215
Contrôle multimédia : `Name`	mmcEnv
Contrôle multimédia : `DeviceType`	WaveAudio
Contrôle multimédia : `PlayEnabled`	True
Contrôle multimédia : `Filename`	\Windows\Media\Chimes.wav
Contrôle multimédia : `Left`	2520
Contrôle multimédia : `Top`	4080
Contrôle multimédia : `Visible`	False
Contrôle multimédia : `Width`	3540

Ajouter le code de la feuille principale

Le Listing PB8.1 contient le code que vous devez ajouter à la feuille principale. Il active la feuille et commande l'animation, qui est simple. Elle résulte du déplacement d'un contrôle PictureBox sur trois endroits de la feuille.

Listing PB8.1 : Le code de l'animation peut être simple

```
1:  Private Sub cmdAni_Click()
2:    ' Utilise le bouton pour commander l'animation
3:    If cmdAni.Caption = "&Animer" Then
4:      cmdAni.Caption = "&Stop"
5:      tmrAni.Enabled = True
```

```
 6:    Else
 7:      cmdAni.Caption = "&Animer"
 8:      tmrAni.Enabled = False
 9:    End If
10: End Sub
11:
12: Private Sub mnuHelpAbout_Click()
13:     mmcEnv.Command = "Open"
14:     mmcEnv.Command = "Play"
15:     frmAbout.Show
16: End Sub
17:
18: Private Sub tmrAni_Timer()
19:   ' Determine le bon emplacement
20:   ' d'image à afficher
21:   '
22:   ' La variable suivante part de zéro
23:   ' et conserve sa valeur à chaque exécution
24:   ' de la procédure.
25:   Static intCounter As Integer
26:
27:   Select Case intCounter
28:     Case 0:
29:       picAni1.Picture = picAni2(1).Picture
30:       picAni2(2).Visible = True
31:       picAni2(2).Left = 3840
32:       picAni2(2).Top = 1220
33:       intCounter = 1
34:     Case 1:
35:       picAni1.Picture = picAni2(1).Picture
36:       picAni2(2).Visible = True
37:       picAni2(2).Left = 4040
38:       picAni2(2).Top = 1120
39:       intCounter = 2
40:     Case 2:
41:       picAni1.Picture = picAni2(1).Picture
42:       picAni2(2).Visible = True
43:       picAni2(2).Left = 4240
44:       picAni2(2).Top = 1220
45:       intCounter = 3
46:     Case 3:
47:       picAni1.Picture = picAni2(0).Picture
48:       picAni2(2).Left = 4440
49:       picAni2(2).Top = 1320
50:       intCounter = 4
51:     Case 4:
52:       ' Arrêter l'animation
53:       picAni1.Visible = True
54:       intCounter = 0
55:       picAni2(2).Visible = False
56:     End Select
57: End Sub
```

Analyse

La procédure événementielle `cmdAni_Click()` bascule le titre du bouton de commande d'Animer à Stop. Si le titre est Animer, la ligne 4 le modifie en Stop et le contrôle Timer est activé à la ligne 5. L'événement `Click` du bouton de commande déclenche alors l'événement `tmrAni_Timer()` de la ligne 18. La propriété `Interval` à 300 signifie que la procédure `tmrAni_Timer()` s'exécutera toutes les 300 millisecondes pour créer l'animation. Elle continue tant que l'utilisateur ne clique pas sur le bouton Stop. La ligne 7 change alors à nouveau le titre en Animer et la ligne 8 désactive le Timer.

La procédure événementielle `tmrAni_Timer()`, qui commence à la ligne 18, est donc exécutée par l'événement Timer toutes les 300 millisecondes. Elle effectue une des cinq actions contrôlées par l'instruction `Select Case` de la ligne 27. La variable de contrôle est une variable statique, déclarée à la ligne 25, qui part d'une valeur de `0` (comme toute variable statique). Une fois une valeur assignée, elle sera préservée lors des exécutions ultérieures de `tmrAni_Timer()`. Elle peut prendre les valeurs de `0` à `4`, modifiées à chaque exécution.

Chacun des trois premiers cas effectue les actions suivantes :

- Assigne à l'enveloppe fermée l'icône de l'enveloppe ouverte (voyez les lignes 29 et 35).
- Rend l'icône de lettre visible (voyez les lignes 30 et 36).
- "Déplace" l'icône de lettre en modifiant ses propriétés `Left` et `Top` (voyez les lignes 31 et 32).
- Incrémente la variable statique `intCounter` pour qu'à l'exécution suivante, un autre groupe d'instructions `Case` s'exécute.

Ce processus se poursuit jusqu'à ce que la valeur de `Case` soit de `4`. A ce moment, à la ligne 53, la procédure montre à nouveau l'enveloppe fermée et masque l'icône de lettre (ligne 55). Sauf si l'utilisateur clique à ce moment sur le bouton Stop, la lettre se remettra à voler hors de l'enveloppe au bout de 300 millisecondes, car la ligne 54 assigne `0` à la variable statique pour obliger l'exécution du premier `Case`.

La procédure événementielle restante, `mnuHelpAbout_Click()`, commande l'affichage de la boîte A propos de à la ligne 15, après que les lignes 13 et 14 ont ouvert et fait jouer le fichier Wav. Les propriétés de la feuille A propos de sont décrites à la section suivante.

Si la boîte A propos de contient du code, ce dernier provient du modèle de feuille utilisé pour la boîte A propos de. Vous n'avez rien à ajouter ou à modifier. Il garantit que la routine Infos système commence lorsque l'utilisateur clique sur le bouton de commande correspondant.

Création de la boîte A propos de

Utilisez le modèle de feuille A propos de pour la boîte A propos de. Une fois la première feuille créée, cliquez du bouton droit dans la fenêtre Projet et sélectionnez Ajouter, Feuille. Sélectionnez A propos de, ce qui ajoute la feuille et le code correspondant au projet.

Le Tableau PB8.2 contient les valeurs de contrôles à utiliser. Le Tableau PB8.2 ne contient que les propriétés à modifier.

Tableau PB8.2 : Configurez ces contrôles et ces propriétés dans la feuille A propos de

Nom de propriété du contrôle	*Valeur de propriété*
lblDescription `Caption`	Voir une animation simple et entendre un son
lblDescription `FontSize`	14
lblDescription `FontStyle`	Gras
lblDisclaimer `Caption`	Attention : programmeur à bord !

Le code qui accompagne la boîte A propos de ajoute le titre et le numéro de version à partir des objets système `App.Title`, `App.Major`, `App.Minor` *et* `App.Revision`. *Ces trois derniers se combinant pour donner le numéro de version 1.0.0.*

Chapitre 18

Interactions avec les données

Ce chapitre vous montre comment accéder aux bases de données à partir de vos applications Visual Basic. Une *base de données* est un regroupement de fichiers qui offre un système de gestion de données complet. Un tel système, tel Microsoft Access, crée la base de données à laquelle votre application Visual Basic peut avoir besoin de se connecter. En utilisant des contrôles et des méthodes spécifiques, vos applications peuvent communiquer avec la base de données.

Vous apprendrez aujourd'hui :

- les bases de données avec lesquelles interagir ;
- la terminologie des bases de donnée ;
- l'importance des champs d'index ;
- les capacités du Gestionnaire des données à analyser les structures de données ;
- le contrôle Data ;
- la comparaison entre les contrôles ADO et le contrôle Data ;
- comment l'assistant Création d'applications de Visual Basic peut analyser les tables et générer des feuilles.

Données de base de données et Visual Basic

En offrant la possibilité d'interagir avec les bases de données, Visual Basic vous permet d'accéder à et de manipuler des ressources de données importantes à partir d'un programme Visual Basic. Les bases de données auxquelles il peut accéder ont des formes et des formats nombreux. Visual Basic supporte les formats de base de données courants suivants :

- Microsoft Access ;
- DBase ;
- FoxPro ;
- bases de données des feuilles de calculs Lotus ;
- bases de données compatibles ODBC ;
- Paradox ;
- fichiers texte avec des données séparées par des virgules.

Une base de données est un regroupement organisé de données, qui ne sont généralement pas basées sur du texte, comme c'est le cas des données de traitement de texte. Ce sont plutôt des groupes d'éléments qu'il faut suivre, que ce soient des personnes, des clients, des fournisseurs, des inventaires, des livres ou des logiciels (en d'autres termes, tout type de données dont vous devez garder la trace, faire des comptes rendus et modifier). Souvent, vous utilisez un système de base de données, tel que Microsoft Access, pour créer et gérer la structure de la base de données. Visual Basic permet d'accéder à la base à partir du système de base de données.

Visual Basic peut accéder aux données de nombreuses versions de ces systèmes de base de données et les gérer. Tant que vous utilisez une version qui existe depuis l'apparition de Windows 95, vous pouvez être sûr que Visual Basic reconnaît son format (mais il supporte aussi certaines versions antérieures à Windows 95).

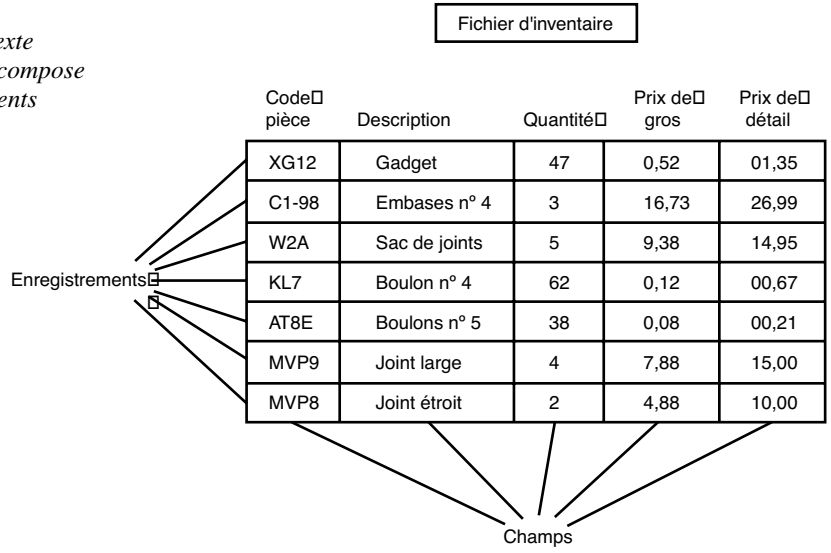
Pour comprendre comment Visual Basic supporte l'utilisation des bases de données, vous devez comprendre les termes apparentés à leur technologie. La leçon d'aujourd'hui ne propose qu'un vernis ! Vous lisez ce livre pour apprendre Visual Basic et pas les bases de données. Pour une approche plus approfondie du sujet, il existe de nombreux ouvrages traitant des bases de données et des accès avancés de Visual Basic.

Il n'est cependant pas utile d'être un expert en bases de données pour connaître les techniques d'accès de Visual Basic. Ce dernier comporte de nombreux contrôles qui gèrent l'accès aux bases de données dans un environnement structuré. Le reste du chapitre vous présente ces outils et le traitement des bases de données en général.

Apprentissage des termes

On commence généralement par apprendre les bases de données en partant des fichiers de données en général. Vous avez déjà un avantage, car vous avez déjà entendu parler des enregistrements et des champs au Chapitre 12. La Figure 18.1 illustre un scénario classique qui montre clairement le concept des enregistrements et des champs. Un enregistrement peut être considéré comme une ligne d'informations dans un fichier de données, même si un enregistrement logique peut s'étendre sur deux ou trois lignes physiques du fichier. Chaque enregistrement est décomposé en champs ou *colonnes*, qui aident à distinguer les éléments de données de l'enregistrement.

Figure 18.1
Un fichier de texte classique se décompose en enregistrements et champs.



Info

Le fichier de données de la Figure 18.1 présente sept enregistrements et cinq champs. Si d'autres éléments sont ajoutés à l'inventaire, le nombre d'enregistrement augmentera, mais le nombre de champs restera le même. Les fichiers de données ne sont cependant pas figés, car vous pouvez augmenter

le nombre de champs. Mais vous devez pour cela changer la structure du fichier ; l'ajout de champs ne constitue habituellement pas le même traitement que l'ajout ou la suppression d'enregistrements. La plupart du temps, vous ajoutez ou retirez des éléments d'inventaire, ce qui revient à ajouter ou supprimer des enregistrements à la base de données.

Les noms de champs ne font jamais partie des données. Ils ne servent qu'à étiqueter les champs, comme les noms de variables étiquettent le contenu des variables.

Le fichier de la Figure 18.1 est dit fichier séquentiel, car il est autonome. Un programme peut accéder simplement aux informations — en séquence ou aléatoirement. Les enregistrements n'ont pas d'ordre particulier, mais le concepteur de la base peut par exemple les classer par numéro de pièce, car ce type de fichier est plus simple à utiliser si on emploie des données triées. Les systèmes de base de données d'aujourd'hui vont bien plus loin que le concept de fichier séquentiel unique illustré par cette figure.

Dans la terminologie de base de données actuelle, une *table* est un fichier de données, et la base de données est le rassemblement de ces tables. Donc, votre base de données peut se composer d'une table des clients, d'une table des fournisseurs, d'une table des employés et d'une table d'inventaire. Lorsque vous les mettez dans une base de données unique, un programme qui a accès à la base a accès à l'ensemble des tables en même temps. Il peut donc déterminer quel fournisseur vous a vendu une pièce en gros ou quel employé a vendu un produit particulier. En d'autres termes, un système de base de données peut prendre ces tables différentes et fournir à partir de là des informations consolidées.

On utilise une *interrogation* pour rechercher des informations dans une base de données. C'est-à-dire qu'un programme interroge la base de données pour rechercher un enregistrement ou un groupe d'enregistrements. Une table de base de données dispose généralement d'au moins un index. Un *index* est un champ de clé avec des valeurs uniques pour chaque enregistrement. L'index fonctionne comme un index de livre : quand vous avez besoin d'accéder à un enregistrement particulier de la table, vous pouvez spécifier la valeur d'index et la base saute directement à la ligne correspondante sans rechercher dans toute la table (comme on le ferait avec des routines classiques d'accès à un fichier séquentiel). Dans la Figure 18.1, le meilleur champ pour l'index serait le champ Code pièce, car chaque pièce a un numéro unique.

Avez-vous déjà eu l'impression de n'être qu'un numéro ? Vous êtes-vous senti fatigué de tous ces numéros présents dans votre vie (compte chèques, compte d'épargne, emprunts, plaques minéralogiques, permis de conduire, numéro de sécurité sociale, etc.) ? Maintenant, vous comprenez mieux l'importance de ces numéros : ils permettent à l'ordinateur de vous identifier

plus rapidement. Des informations enregistrées par noms seraient plus difficiles à retrouver. Par exemple, plusieurs personnes dans un fichier national de cartes de crédit auront probablement le même nom que vous. Si vous vous appelez Delabrosse et que vous demandez votre solde actuel, le réceptionniste pourrait chercher De La Brosse, ou Labrosse ou DELABROSSE, et l'ordinateur ne pas retrouver le nom. Les machines sont tellement bornées ! L'identifiant numérique unique utilisé pour les champs d'index signifie que moins d'erreurs se produiront. Les entreprises peuvent également automatiser pour gagner du temps, ce qui se ressent finalement dans de meilleurs prix et taux d'intérêts.

Lorsqu'ils travaillent sur des bases de données, les programmeurs préfèrent le terme *table* à celui de *fichier*. Cependant, ils utilisent également *colonnes* et *lignes* pour désigner les *champs* et les *enregistrements*. C'est judicieux, car les fichiers, ou *tables*, sont en théorie rectangulaires (mais pas en réalité, car ils sont enregistrés physiquement d'une manière différente) et se composent de colonnes et de lignes (voir Figure 18.1). Parler de plusieurs fichiers dans un seul fichier de base de données peut également prêter à confusion, de là le terme table pour une occurrence d'un ensemble de données (un fichier de données) dans la base de données. En outre, la plupart des systèmes de base de données actuels sont *relationnels*, ce qui signifie qu'il n'y a pas deux tables contenant exactement les mêmes données pour éviter autant que possible la redondance des fichiers. Microsoft Access est de ce type. Les fichiers de base de données non relationnels, comme les fichiers dBase d'avant la version 4.0, demandent l'ajout de code Visual Basic assez complexe pour que le fichier imite les accès relationnels avant de pouvoir effectuer des Entrées/Sorties avec les outils de base de données de Visual Basic.

La plupart des bases de données supportent également l'interface utilisateur. Par exemple, les fichiers de base de données peuvent contenir des états qui génèrent des sorties en fonction des données, des formulaires écrans pour afficher et recevoir les nouvelles données des utilisateurs, des interrogations enregistrées pour éviter à l'utilisateur de créer une nouvelle requête chaque fois qu'il a besoin d'informations, et des définitions de bases de données pour que les programmes puissent analyser la base de données et lire le format à l'aide de procédures standards. Ces procédures permettent aux programmeurs de savoir combien de tables existent et à quoi elles ressemblent.

Obtention d'un échantillon de données

Visual Basic est fourni avec les deux bases de données exemples, toutes deux au format Microsoft Access :

- BIBLIO.MDB, qui contient une base de données de vendeurs et de titres de livres informatiques.
- NWIND.MDB, qui contient le système de base de données complet d'une entreprise imaginaire, comprenant l'inventaire, les clients, les fournisseurs, les employés, les statistiques des ventes, etc. Le nom de l'entreprise est Northwind Traders.

Tous les noms de fichiers des bases de données Access se terminent par .MDB (pour *Microsoft Database*). Un fichier de base de données Access peut être volumineux, car il contient l'ensemble des tables, états, formulaires, écrans et requêtes enregistrés dans la base de données. L'avantage du fichier unique est une simplification de la sauvegarde de la base complète (il n'est pas utile de garder la trace de plusieurs fichiers lorsqu'on effectue la sauvegarde).

Pour que cette présentation des bases de données reste simple, la présente leçon utilise la base de données NWIND.MDB pour illustrer l'utilisation de certains contrôles et commandes liés aux bases de données. Cependant, tous les programmeurs Visual Basic n'ont pas accès à un système de base de données externe. Heureusement, Visual Basic comporte un outil complémentaire spécial nommé le Gestionnaire de données qui vous permet de créer et de modifier des fichiers de base de données.

Le Gestionnaire de données est un programme de complément, disponible à partir de l'environnement Visual Basic, que vous pouvez utiliser pour créer des bases de données, entrer modifier des données, et modifier et rendre compte de leur structure. Les fichiers créés et analysés dans le Gestionnaire de données vous aident à écrire et à tester des programmes Visual Basic qui doivent travailler sur des données identiques.

Si vous utilisez la version standard de Visual Basic, vous ne trouverez pas le Gestionnaire des données, mais la base de données NWIND.MDB est toujours là, ce qui vous permet de travailler sur certains des exemples qui suivent.

Le Gestionnaire de données est le seul outil par défaut du menu Compléments. Sélectionnez Compléments, Gestionnaire de données pour que Visual Basic démarre le programme (voir Figure 18.2).

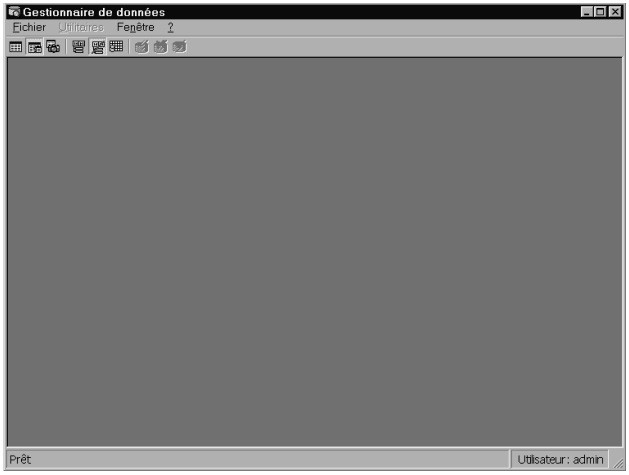


Figure 18.2
Le Gestionnaire de données vous aide à créer et à analyser des fichiers de base de données.

Le Gestionnaire de données vous permet d'effectuer les tâches suivantes :

- créer de nouvelles bases de données ;
- entrer de nouvelles données dans les fichiers de données ;
- modifier les structures des fichiers de base de données existants ;
- modifier les données des fichiers de base de données existants ;
- rechercher des champs, des requêtes et des jeux d'enregistrements.

Définition

Un jeu d'enregistrements (recordset) est simplement un regroupement d'enregistrements. Il en existe plusieurs types. Le type par défaut est celui constitué par l'ensemble des enregistrements (lignes) d'une table. Vous pouvez créer un nouveau jeu d'enregistrements en ne recherchant que les enregistrements qui répondent à un critère particulier (par exemple, "Tous les enregistrements dont le champ Solde est supérieur à 3 000 F"). Un dynaset est un jeu d'enregistrements qui se modifie, mais en continuant à répondre à un certain critère lorsque vous l'interrogez. Un snapshot est un jeu d'enregistrements pris à un instant précis, par exemple tous les enregistrements qui répondent à un critère particulier le dernier jour du mois.

En d'autres termes, le Gestionnaire de données se comporte comme un système de données, pas très différent de Microsoft Access. Mais n'oubliez cependant pas qu'il est très limité et que c'est plus un outil d'administration pour l'analyse de fichiers de base de données qu'un véritable système de base de données. Vous ne pouvez par exemple pas créer d'états ni de formulaires.

Rappelons que cette leçon ne présente qu'un survol rapide de la technique et de la terminologie des bases de données. La maîtrise d'un système de base de données complet tel que Microsoft Access prend autant de temps sinon plus que celle du système de développement Visual Basic.

Voici les étapes générales pour créer une nouvelle base de données avec le Gestionnaire de données de Visual Basic :

1. Créez le fichier de base de données en sélectionnant Fichier, Nouvelle base de données, puis en choisissant le type de base que vous voulez créer en choisissant dans la liste. Le Gestionnaire de données peut créer et modifier tous les formats de base de données listés au début de cette leçon.
2. Créez chaque table de la base en cliquant du bouton droit sur la fenêtre Base de données et en sélectionnant Nouvelle table dans le menu contextuel. Le Gestionnaire de données affiche la feuille Modifier la structure, illustré à la Figure 18.3, dans lequel vous pouvez entrer le nom de la table, chacun des champs, leur type (chaque champ de base de données supporte un type de données unique, comme les variables dans Visual Basic), les exigences de sécurité et de validation (par exemple les champs protégés par mots de passe), et les champs indexés (une table peut avoir plusieurs index suivant la manière dont l'application accède aux données).

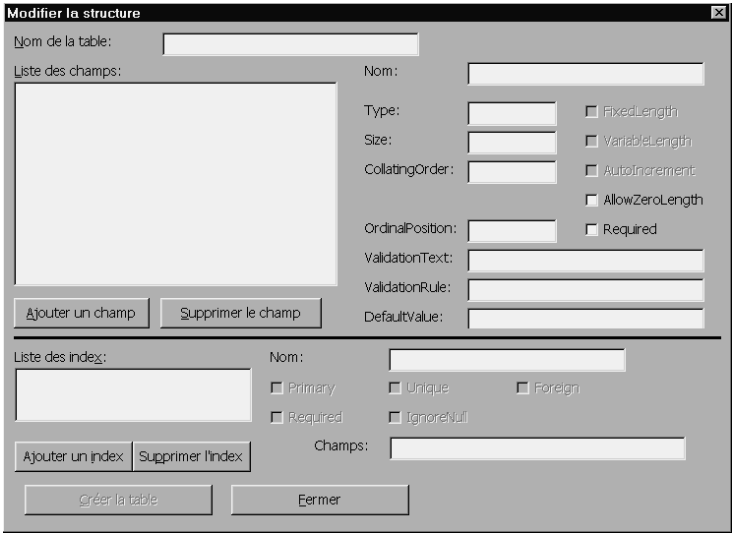


Figure 18.3
Vous pouvez concevoir chaque table et la structure de ses champs à partir du formulaire Modifier la structure.

3. Cliquez sur le bouton Créer la table pour créer la structure de la table et pour ajouter des tables supplémentaires.
4. Cliquez sur le bouton de la barre d'outils dont l'info-bulle annonce "Utiliser le contrôle Data sur la feuille" pour que le Gestionnaire de données de Visual Basic ait un outil (le contrôle Data) à utiliser lors de la saisie des données de table de la base de données.
5. Double-cliquez sur un des noms de table de la fenêtre de base de données et entrez chaque enregistrement en utilisant le formulaire de saisie illustré à la Figure 18.4.

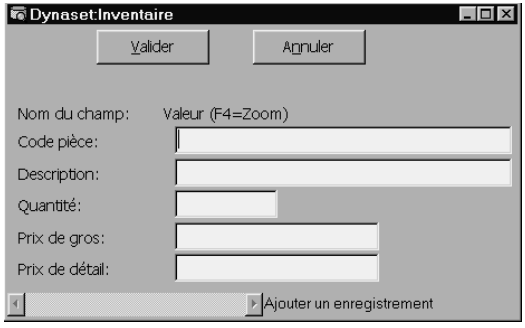


Figure 18.4
Entrez les données de chaque table pour avoir une base de données sur laquelle travailler.

Si vous avez d'abord cliqué sur le bouton de la barre d'outils de jeu d'enregistrements de type dynaset, les données que vous allez saisir prendront la forme d'un de ces types avancés de jeux d'enregistrements.

6. Sélectionnez Fichier, Fermer pour fermer et enregistrer la nouvelle base de données.

Bien que la création d'une base de données exige bien plus de détails que ce qu'il en est dit ici, vous avez désormais une idée générale des étapes nécessaires à sa création en n'utilisant que les outils fournis par Visual Basic.

Si votre programmation en arrive à exiger un travail important sur les bases de données, il vaut mieux ne pas vous fier au Gestionnaire de données de Visual Basic pour générer et modifier vos fichiers de base de données. S'il est utile pour décrypter les formats de base de données et pour créer des bases simples de test et de débogage, ses fonctionnalités sont très loin de celles de Microsoft Access, FoxPro, ou d'autres systèmes du marché. Par exemple, n'utilisez pas le Gestionnaire de données pour concevoir et créer la base de données complète d'une entreprise. Il n'est pas assez souple ni simple pour travailler régulièrement à la gestion d'une base de données.

Pour une programmation sérieuse de base de données, il est préférable d'ajouter un système de gestion de base de données à votre collection d'applications.

Même si le Gestionnaire de données de Visual Basic offre quelques outils d'interrogation permettant de rechercher des données et créer des requêtes complexes pour trouver des enregistrements, il reste limité à l'administration de la base de données. Visual Basic attend patiemment de vous permettre de créer une application complète pour réellement travailler sur une base de données. Maintenant que vous avez eu une présentation du Gestionnaire de données, vous êtes prêt à voir comment Visual Basic peut accéder aux fichiers de la base de données que vous avez créés, avec le Gestionnaire de données, ou avec un système autonome de gestion de base de données tel que dBase. Comme Visual Basic est fourni avec le fichier de base de données NWIND.MDB, il est inutile de créer une base pour suivre la leçon d'aujourd'hui.

Le contrôle Data

Le contrôle Data, ou contrôle de données, dernier contrôle intrinsèque de la fenêtre Boîte à outils qu'il reste à étudier, est souvent considéré comme un outil lent et encombrant de gestion des données. Mais, pour sa défense, il est toujours utile pour les raisons suivantes :

- C'est un contrôle simple, qui rend plus aisé l'apprentissage de la manière dont Visual Basic interagit avec les fichiers de base de données.
- Le contrôle Data est toujours présent dans la fenêtre Boîte à outils. Il n'est donc pas nécessaire de chercher et d'ajouter un contrôle ActiveX pour l'utiliser.
- L'édition Standard de Visual Basic ne contient pas tous les outils avancés de base de données inclus dans les éditions Professionnelle et Entreprise. Dans la partie finale de la leçon d'aujourd'hui, vous apprendrez comment ces outils et techniques avancés peuvent aider le programmeur à accéder aux bases de données de manière plus variée que ne le permet le contrôle Data. Mais tout le monde ne possédant pas ces versions, ces outils plus avancés ne sont pas disponibles pour tous, alors que le contrôle Data l'est.

Les sections qui suivent décrivent comment utiliser le contrôle Data pour accéder à la base de données NWIND.MDB. Une fois celui-ci maîtrisé, les contrôles de données plus avancés ne vous paraîtront plus si impressionnants.

Configurer le contrôle Data

La méthode la plus simple pour accéder à une base de données est d'utiliser le contrôle Data. Quand vous le placez sur une feuille, il ressemble un peu au contrôle multimédia étudié au jour 14. Si vous l'élargissez quelque peu, vous constaterez qu'il ressemble à celui que montre la Figure 18.5.

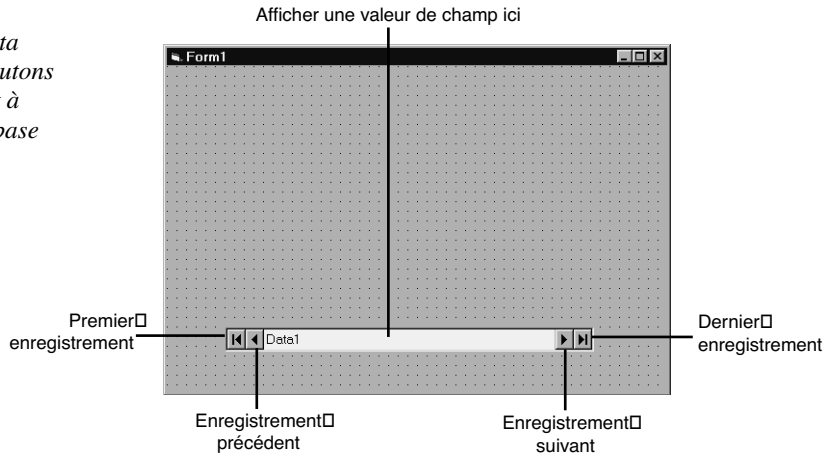


Figure 18.5
Le contrôle Data contient des boutons qui vous aident à parcourir une base de données.

Un contrôle Data se compose de :

- Deux flèches intérieures qui permettent d'avancer et de reculer d'un enregistrement à la fois dans une table de base de données.
- Des flèches extérieures qui vous conduisent au premier ou au dernier enregistrement de la table de base de données.
- Une zone centrale qui affiche l'information que vous voulez de la base de données.

Le contrôle Data est un *contrôle lié*. Vous pouvez lier de nombreux contrôles Visual Basic, tels que le contrôle TextBox, à une base de données. Quand un utilisateur parcourt la base, le champ de texte affiche le champ configuré. Lorsque vous liez un contrôle à une base de données, vous n'avez plus à vous préoccuper de l'affichage des données du champ ; c'est Visual Basic qui s'occupe de tout.

Un contrôle lié est un contrôle lié aux données de votre base de données de manière à ce qu'il en rende le parcours simple pour un programme Visual Basic.

Pour afficher un enregistrement à la fois à partir d'un jeu d'enregistrements que vous avez défini dans la base de données, utilisez un contrôle lié. Il permet de n'afficher en général que l'enregistrement en cours ou un de ses champs. Lorsque l'utilisateur navigue dans la base en cliquant sur les boutons du contrôle Data, l'enregistrement en cours change et reflète la position de l'utilisateur dans la table de la base de données.

Utiliser le contrôle Data

Pour voir fonctionner le contrôle Data, prenez le temps de créer une application qui affiche des enregistrements d'une table de la base NWIND.MDB. Les étapes suivantes en décrivent le processus :

1. Créez une nouvelle application et ajoutez le contrôle Data à la feuille. Modifiez la valeur de propriété `Width` à `4620` pour laisser de la place au texte à afficher dans la partie centrale du contrôle. Mettez le contrôle en bas de la fenêtre en donnant à la propriété `Top` la valeur `3240`.
2. Nommez le contrôle `dtaBooks`.
3. Utilisez la propriété `DatabaseName` pour connecter le contrôle à la base de données. Double-cliquez sur la propriété pour sélectionner le fichier NWIND.MBD dans le dossier Visual Basic. Le contrôle Data est un contrôle intelligent qui sait distinguer les différentes tables d'une base de données. Il permet de n'accéder qu'à une table à la fois (ou plus précisément à un jeu d'enregistrements à la fois) et, une fois la propriété `DatabaseName` configurée, le contrôle est capable de déchiffrer les tables de la base.
4. Double-cliquez sur la propriété `RecordSource` et sélectionnez la table Customers dans la liste. La propriété `RecordSource` spécifie le jeu d'enregistrements géré par le contrôle Data lorsque l'application accède à la base.
5. Modifiez la propriété `Caption` du contrôle Data en `Cliquez pour changer de client`. Le texte s'affiche dans la zone centrale du contrôle.
6. Ajoutez à la feuille un contrôle étiquette avec les propriétés suivantes :

```
Name: lblCust
Alignment: 2-Center
Font.Size: 18
Height: 915
Left: 1680
Top: 1440
Width: 4320
```

7. Ouvrez la propriété `DataSource` de l'étiquette. C'est une propriété que vous n'avez encore jamais eu à configurer. Vous allez lier l'étiquette au contrôle Data. (Une étiquette est un des contrôles qui peuvent être liés). La propriété `DataSource` vous

permet de choisir la valeur de `dtaBooks`, car c'est la table de la base de données à laquelle le contrôle Data est rattaché. S'il y a plusieurs contrôles Data sur la feuille et qu'ils sont déjà attachés à des bases de données, la propriété `DataSource` les listerait tous, pour vous permettre de sélectionner la source de cette étiquette de données.

8. L'étiquette étant liée à la bonne table, il reste à la lier à la bonne colonne (*champ*). Ouvrez la propriété `DataField` et sélectionnez `ContactName` comme colonne de la table à afficher.
9. Vous êtes prêt à exécuter l'application. Appuyez sur F5 pour cela : le nom du premier client s'affichera automatiquement dans l'étiquette. N'oubliez pas qu'elle est liée au contrôle Data, lui-même lié à la base de données. Quand vous cliquez sur les boutons du contrôle Data, le contenu du champ change et reflète le client en cours (voir Figure 18.6)
10. Cliquez sur le bouton de fermeture de la fenêtre de l'application pour terminer le programme et enregistrer votre projet..

Figure 18.6
Vous pouvez maintenant parcourir la base de données, enregistrement par enregistrement, en avant ou en arrière.

Vous n'êtes absolument pas limité à une colonne de la table. Vous pouvez ajouter autant d'étiquettes que vous le voulez pour afficher plusieurs colonnes. La Figure 18.7 montre l'affichage d'un enregistrement client complet. Les étiquettes supplémentaires ont simplement leur valeur `DataField` liée à des colonnes particulières de la table. En outre, des étiquettes supplémentaires aident à la description des données affichées. (Vous pourrez modifier le programme de base de données en ajoutant ces colonnes supplémentaires dans le premier exercice à la fin du chapitre).

Pour afficher les colonnes de plusieurs tables, vous devez créer un jeu d'enregistrements, tel qu'un dynaset, dans le système de base de données pour extraire les colonnes des tables correctes. Des commandes et des contrôles Visual Basic plus avancés vous permettent d'effectuer des sélections complexes dans les bases de données, comme vous le verrez dans les dernières parties de la leçon d'aujourd'hui.

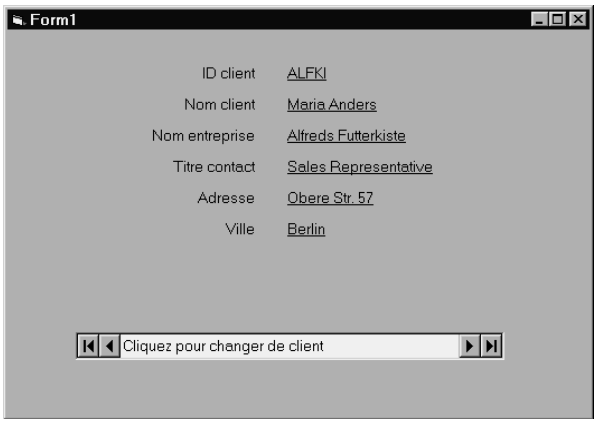


Figure 18.7
Afficher plusieurs colonnes de tables ne demande que des étiquettes supplémentaires.

Pour afficher les données de type booléen, utilisez une case à cocher ou un bouton d'option pour indiquer les conditions `True` *ou* `False` *du champ. L'option peut indiquer les valeurs de données Oui/Non ou Vrai/Faux qui apparaissent dans les tables des bases de données.*

Utilisation avancée du contrôle Data

Les applications de base de données ne doivent pas obligatoirement se cantonner à la lecture. Vous pouvez souhaiter laisser l'utilisateur modifier les données. Il existe plusieurs manières de le faire. L'approche la plus simple consiste à afficher les données dans des zones de texte au lieu d'étiquettes. Les utilisateurs verront les informations exactement comme avec les étiquettes, mais ils pourront aussi modifier les informations dans la zone de texte pour mettre à jour la base.

Soyez toujours vigilant sur la sécurité. Vous pouvez avoir besoin d'afficher une boîte de dialogue de saisie de mot de passe pour limiter la mise à jour à certaines personnes. De plus, la base elle-même peut limiter les types de modifications autorisées. La propriété `Exclusive` *du contrôle Data*

détermine si un utilisateur peut avoir des droits exclusifs sur une base de données, et être le seul utilisateur autorisé à y accéder (dans les systèmes en réseau) ; sinon, le contrôle Data n'offre que peu en matière de sécurité. Vous devrez maîtriser des commandes et des contrôles plus avancés (dont certains sont étudiés plus loin) pour incorporer une véritable sécurité dans vos applications.

Lorsque votre savoir-faire en programmation s'améliorera, vous apprendrez des méthodes d'utilisation du contrôle Data permettant un accès plus avancé aux bases de données. Il dispose par exemple de plusieurs méthodes `Move` qui permettent de déplacer le pointeur d'enregistrement comme le font les boutons du contrôle Data.

Les méthodes suivantes déplacent le pointeur au premier ou au dernier enregistrement, au précédent ou au suivant, dans la base de données pointée par la propriété `DataSource` du contrôle Data :

```
dtaCust.Recordset.MoveFirst     ' Va au premier enregistrement
dtaCust.Recordset.MoveLast      ' Va au dernier enregistrement
dtaCust.Recordset.MoveNext      ' Enregistrement suivant
dtaCust.Recordset.MovePrevious  ' Enregistrement précédent
```

Le pointeur d'enregistrement garde la trace de l'enregistrement en cours dans la table de la base de données ouverte. Lorsque vous ouvrez une base, le pointeur indique le premier enregistrement. Lorsque vous lisez en séquence la table, le pointeur d'enregistrement avance. Les méthodes de type `Move` manipulent le pointeur et permettent d'accéder à différents enregistrements de la table.

Le jeu d'enregistrements par défaut du contrôle Data est défini par les valeurs de propriétés que vous configurez. Par exemple, si vous ajoutez un bouton de commande qui comporte une procédure événementielle `Click` qui contient à son tour une de ces méthodes, l'étiquette affiche l'enregistrement sélectionné par la méthode chaque fois que l'utilisateur clique sur le bouton de commande. Outre les contrôles de déplacement dans les enregistrements, il existe des méthodes qui ajoutent ou suppriment des enregistrements de la base de données.

Utilisez les propriétés booléennes `BOF` et `EOF` pour voir si le pointeur d'enregistrement se trouve au début ou à la fin d'une table. Les programmeurs Visual Basic utilisent souvent une boucle `Do…While` pour parcourir tous les enregistrements de la table. La boucle se termine quand `datCust.Recordset.EOF` est égale à `True`.

Contrôles avancés de base de données

Les éditions Professionnelle, Entreprise et Visual Studio de Visual Basic disposent d'un ensemble avancé de contrôles, propriétés, méthodes et événements pour écrire des applications importantes d'accès aux bases de données. Si ce cours en 21 jours ne peut pas explorer tous les concepts avancés des bases de données, cette leçon en offre une présentation. Si vous devez ajouter à vos applications une meilleure gestion de base de données, vous devez au moins savoir de quoi est capable Visual Basic et mieux comprendre une part de la terminologie.

Depuis sa version 6, Visual Basic supporte une grande diversité d'objets *ADO* (ADO signifie *ActiveX Data Objects*, ou Objets de données ActiveX). Etants basés sur ActiveX, ces objets fonctionnent sur différentes plates-formes et avec différents langages de programmation (contrairement au contrôle Data qui est strictement limité à l'environnement Visual Basic). Les objets ADO supportent l'accès aux bases de données pour un objet de données local ou distant (*RDO — remote data object*). Les données distantes peuvent provenir d'un réseau ou d'une ligne de communication.

La maîtrise des contrôles ADO est importante, car ils offrent plusieurs avantages par rapport au contrôle Data. Malgré la formation qu'ils demandent (il reste beaucoup à apprendre pour pouvoir les utiliser pleinement), ils sont le choix habituel chez les programmeurs Visual Basic de bases de données, du fait de leur puissance et de leur souplesse.

La technologie ADO supporte des accès aux bases de données plus rapides qu'avec le contrôle Data. Les ordinateurs actuels sont rapides, mais vous observerez une dégradation quand vous utilisez le contrôle Data avec de grosses bases de données, en particulier sous OBDC.

En utilisant ADO, vous aurez sans doute à écrire plus de code qu'avec le contrôle Data. Même si vous pouvez écrire du code d'accès aux diverses méthodes du contrôle Data, l'accès direct à la base de données est moins compliqué. ADO permet de contrôler l'accès aux données d'une manière bien plus stricte que le contrôle Data, dont la simplicité reflète le manque de flexibilité. La surcharge causée par le contrôle Data ne grève pas les programmes utilisant ADO.

L'avantage principal d'ADO est sans doute sa capacité à accéder à de nombreux types de données. Ne se limitant pas aux seules informations des bases de données, relationnelles ou pas, les contrôles ADO peuvent accéder, par une programmation pointue, aux navigateurs Internet, au courrier électronique, et même à des graphiques.

Plusieurs contrôles ADO placés sur une même feuille peuvent dégrader les performances d'une application, car chacun fonctionne indépendamment des autres et consomme son propre ensemble de ressources.

La technologie ADO supporte (dans toutes les éditions dans une certaine mesure, mais complètement dans les éditions Professionnelle, Entreprise et Visual Studio) les contrôles de données suivants :

- **Contrôle Data ADO**. Il fonctionne comme un contrôle Data en se connectant à une base de données et en permettant à l'utilisateur de parcourir les enregistrements.
- **Contrôle DataCombo ADO**. Il ressemble à une zone de liste déroulante standard, mais donne à l'utilisateur l'accès à plusieurs enregistrements de cette colonne.
- **Contrôle DataList ADO**. Il ressemble à une zone de liste standard, mais donne à l'utilisateur l'accès à plusieurs enregistrements de cette colonne.

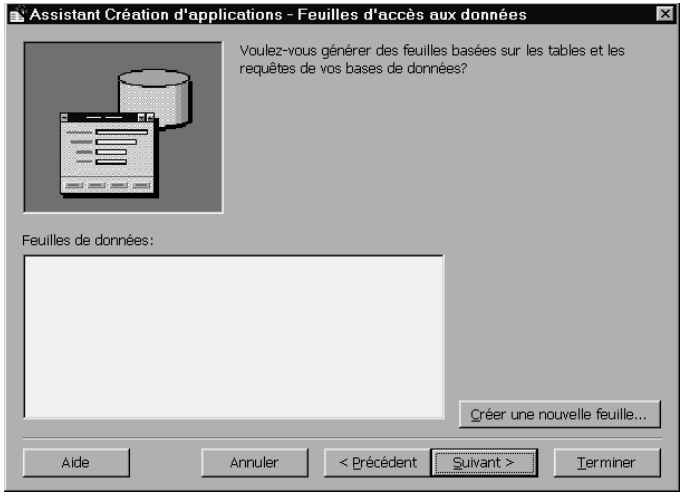
Des versions non ADO des ces contrôles sont disponibles.

L'assistant Création d'applications

Visual Basic peut effectuer une bonne partie du travail si vous utilisez l'assistant Visual Basic Création d'applications pour générer une application contenant un accès à la base de données. Bien que l'assistant fournisse un accès plus limité que celui que permet la programmation ADO, vous pouvez créer l'application initiale, puis la modifier pour générer une application plus complète.

Le code résultant de la génération de l'application par l'assistant est assez complet et il forme la base d'une véritable application de base de données. La Figure 18.8 montre la première fenêtre de l'assistant consacrée à l'accès à la base de données.

Figure 18.8
L'assistant Création d'application vous permet de créer des programmes de base de données.



Supposons que vous vouliez une application qui propose un accès en consultation et en modification au fichier de base de données BIBLIO.MDB. Après avoir lancé l'assistant Création d'application et passé plusieurs fenêtres, vous verrez la fenêtre de la Figure 18.8. Vous y demandez à l'assistant de générer les feuilles d'accès à la base de données. Vous pourriez les créer par vous-même, mais vous pouvez aussi laisser faire l'assistant, puis personnaliser ensuite les feuilles.

Une fois que vous cliquez sur le bouton Créer une nouvelle feuille, l'assistant commence par vous demander un profil. Dans cet exemple, cliquez sur Suivant pour sélectionner un type de base de données pour l'application. La base BIBLIO .MDB fournie avec Visual Basic étant une base Microsoft Access, sélectionnez Access, puis cliquez sur Suivant.

L'assistant demande ensuite le nom de la base de données dans la fenêtre illustrée à la Figure 18.9. Spécifiez le chemin d'accès et le nom de la base. (Vous pouvez cliquer sur le bouton Parcourir pour chercher le fichier.)

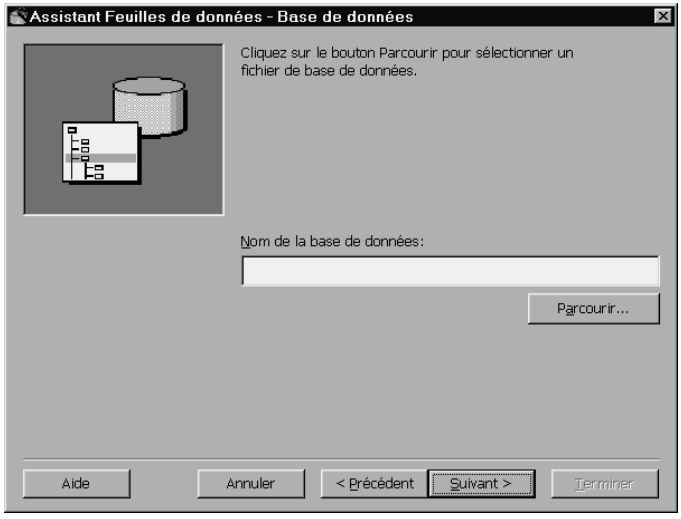


Figure 18.9
L'assistant demande la base de données à utiliser.

Une fois la base spécifiée, cliquez sur Suivant ; l'assistant demande le nom du formulaire principal (entrez frmADO dans cet exemple) et la présentation que vous voulez. Vous avez les cinq choix de présentation suivants :

- **Enregistrement unique**. L'utilisateur ne peut qu'accéder à un enregistrement à la fois, en consultation ou modification.

- **Grille (feuille de données)**. L'utilisateur peut accéder à plusieurs enregistrements à la fois dans un affichage de type table, en consultation ou modification.
- **Principale/secondaire**. L'utilisateur peut accéder, en consultation ou modification, à des enregistrements secondaires apparentés à un enregistrement principal unique ; par exemple, tous les produits (les enregistrements secondaires) qu'un fournisseur (l'enregistrement principal) a acheté dans le passé, au travers d'une *relation un-à-plusieurs*.
- **MS HFlexGrid**. L'utilisateur peut accéder à plusieurs enregistrements dans un format de table, en consultation ou modification.
- **MS Chart**. L'utilisateur peut accéder à plusieurs enregistrements dans un format de diagramme, en consultation ou modification.

Une relation un-à-plusieurs existe entre des enregistrements de plusieurs tables dans la plupart des bases de données. Un enregistrement peut contenir une valeur de colonne qui se retrouve sur plusieurs enregistrements d'un autre fichier. Par exemple, une base de données d'auteurs contiendra une table des auteurs. Une table des ouvrages peut exister avec plusieurs titres écrits par le même auteur. Ce dernier apparaît comme l'enregistrement principal, tandis que les livres sont les enregistrements secondaires de cette relation auteur à livre, de un à plusieurs.

Vous pouvez aussi spécifier les *liens* (la manière dont Visual Basic lie les informations de la base de données aux contrôles). L'assistant propose ces trois liens de données :

- **Contrôle de données ADO**. Utilise le contrôle de données ADO pour lier les contrôles aux données.
- **Code ADO**. Utilise du code ADO pour lier les contrôles aux données.
- **Classe**. Crée une classe de données particulière pour la base de données et lie les données aux contrôles au travers de cette classe.

Pour cet exemple, sélectionnez Contrôle de données ADO et cliquez sur Suivant. Dans la fenêtre suivante, l'assistant doit connaître la table, ou le jeu d'enregistrements, auquel accéder dans la base. Ouvrez la liste déroulante Source d'enregistrements pour sélectionner la table Publishers. Comme le montre la Figure 18.10, la liste des champs disponibles est mise immédiatement à jour ; elle affiche tous les champs, ou *colonnes*, de la table.

Les champs que vous sélectionnez déterminent ceux que l'assistant place sur la feuille ADO qu'il génère. Vous pouvez sélectionner un champ de la liste de gauche et cliquer sur le bouton > pour l'envoyer dans la liste de droite, qui sert à générer les champs de la feuille finale. Dans cet exemple, envoyez-y tous les champs.

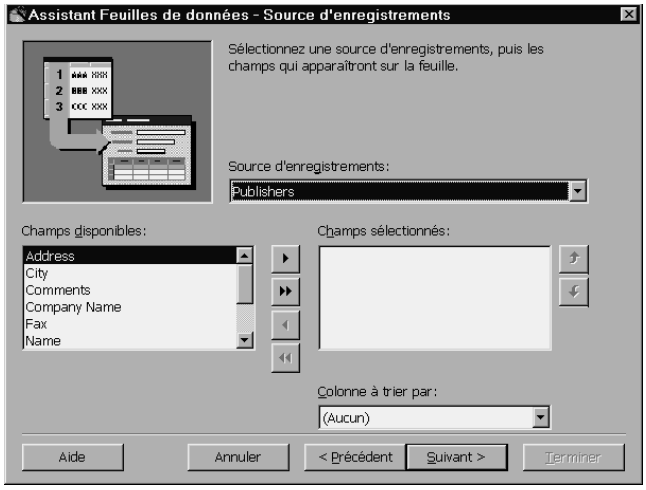


Figure 18.10
Visual Basic analyse la table et affiche les champs que vous pouvez sélectionner.

Réorganisez l'ordre des champs pour que `Company Name` apparaisse en premier. Pour cela, cliquez sur ce champ, puis cliquez sur la flèche vers le haut trois fois pour déplacer le champ en tête de liste. Pour que la feuille affiche les enregistrements par ordre alphabétique d'entreprise, sélectionnez `Company Name` dans la liste des colonnes à trier.

Cliquez sur le bouton Suivant pour afficher la fenêtre de sélection des contrôles illustrée à la Figure 18.11. Cette fenêtre vous permet de spécifier les boutons qui s'afficheront dans la feuille d'accès aux données. Ils reflètent les capacités que vos accordez à l'utilisateur. Pour l'empêcher de supprimer des enregistrements, décochez le bouton d'option Supprimer. Dans l'exemple, gardez tous les boutons cochés.

Lorsque vous cliquez sur le bouton Terminer, Visual Basic génère l'application.

Le bouton Terminer n'achève pas l'assistant, mais uniquement la partie création de feuille. Vous devez achever l'assistant normalement. Dans cet exemple, en revenant de la génération de feuille, vous pouvez cliquer sur le bouton Terminer.

Lorsque vous exécutez l'application, la feuille Publishers s'affiche (voir Figure 18.12.) L'assistant y a placé tous les boutons et zones de texte nécessaires pour que les champs de données puissent être accessibles correctement. Vous pourrez ajouter des éléments à la feuille, peut-être la personnaliser pour la rendre plus attrayante, une fois que vous avez vérifié que l'application générée fonctionne correctement.

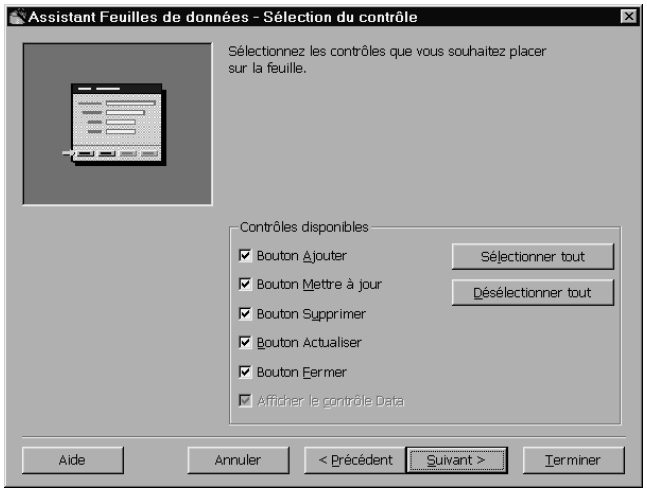


Figure 18.11
Vous pouvez contrôler la capacité de l'utilisateur à ajouter, modifier et supprimer des enregistrements de la table de base de données.

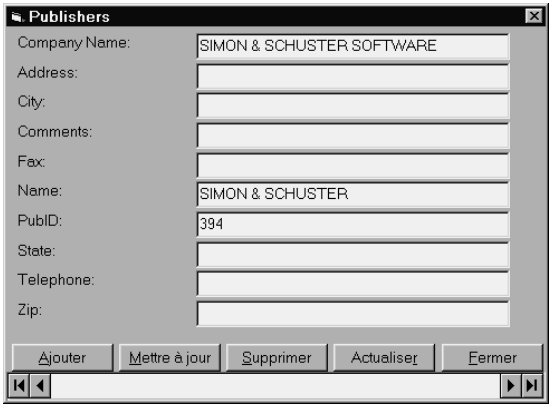


Figure 18.12
L'assistant a généré la feuille avec tous les champs et les boutons de base de données nécessaires.

En résumé

La leçon d'aujourd'hui a traité de l'accès aux bases de données à l'aide de Visual Basic. La manière la plus simple d'accéder à une base de données (en considérant que vous n'utilisez pas l'assistant Création d'applications) est d'ajouter le contrôle Data à votre feuille. Il s'occupe de mettre à jour la base de données sous-jacente et de modifier les contrôles liés lorsque vous vous déplacez dans les enregistrements. Des méthodes peuvent être utilisées pour étendre les fonctionnalités du contrôle Data.

L'interface ADO peut exiger une programmation complexe, mais vous aurez beaucoup plus de contrôle et de souplesse dans l'accès à la base de données. Même si votre application doit s'occuper de mettre à jour les contrôles et le déplacement entre les enregistrements lorsque les événements sont déclenchés, l'application est globalement plus rapide.

La leçon de demain explore des manières d'intégrer l'Internet à Visual Basic. Le monde informatique est en train de devenir rapidement un monde connecté et vos applications ont souvent besoin de proposer des fonctionnalités Internet.

Questions-réponses

Q Les relations un-à-plusieurs existent-elles toujours dans les tables de bases de données ?

R Il existe plusieurs types de relations entre les données, qui peuvent se retrouver toutes ou en partie dans une base de données, en fonction de sa structure. Rappelons que le but du chapitre n'est pas d'approfondir la théorie des bases de données. Cependant, les relations un-à-plusieurs étant nécessaires dans de nombreuses applications Visual Basic, une meilleure compréhension de ce sujet n'est pas un mal. Vous aurez souvent à afficher tous les enregistrements en rapport à une valeur de clé donnée, c'est ce qu'offre cette relation.

L'affichage Principal/Secondaire de l'assistant est une excellente manière de créer des applications qui offrent cette relation entre les données. Il faut au moins un champ en correspondance entre les deux tables pour que la relation soit possible. Une relation *un-à-un* se présente parfois, lorsqu'une colonne d'une table est apparentée à une autre colonne dans une autre table. Ce peut être le cas par exemple d'une même pièce proposée par deux fournisseurs. Des relations *plusieurs-à-plusieurs* existent également. Toutes ces relations sont définies lors de la conception et elles sont la base de la compréhension du fonctionnement des bases de données relationnelles. Le principal dans ces relations est la manière dont on y accède. Votre tâche de programmeur Visual Basic consiste à accéder à ces relations, mais vous n'aurez pas à les reconstruire, sauf si vous être le concepteur et le créateur de la base de données.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités et des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se trouvent à l'Annexe A

Quiz

1. Quel est l'outil fourni avec Visual Basic permettant la modification et la consultation des fichiers de base de données ?
2. Quelle est la différence entre un fichier et une table ?
3. Vrai ou faux. Lorsque vous ajoutez des enregistrements à une table, le nombre de ses colonnes augmente également.
4. Vrai ou faux. Une table est un sous-ensemble d'un jeu d'enregistrements.
5. Qu'est-ce qu'un contrôle lié ?
6. Quelles sont les différences entre jeux d'enregistrements recordset, dynaset et snapshot ?
7. Donnez deux avantages d'ADO sur le contrôle Data ?
8. Que définissent les valeurs `EOF` et `BOF` ?
9. Quelle est la différence entre un affichage Principal et un affichage Secondaire ?
10. Quel est l'outil fourni avec Visual Basic pour générer des feuilles directement à partir de la structure de la base de données ?

Exercices

1. Modifiez l'application d'accès à la base de données que vous avez créée (illustrée à la Figure 18.6), pour qu'elle affiche tous les champs de la table Customer. Votre feuille doit imiter celle de la Figure 18.7. (N'oubliez pas d'ajouter les étiquettes descriptives pour que l'utilisateur sache ce que contient chaque colonne.)
2. Utilisez l'assistant Création d'application pour générer un affichage Principal/Secondaire de la base de données BIBLIO.MDB qui présente le nom de l'auteur en affichage principal et tous les codes ISBN de ses ouvrages dans l'affichage secondaire.

PB9

Contrôles ADO

Ce projet décrit comment créer une application de base de données sous ADO. Vous utiliserez la base de données exemple BIBLIO.MDB fournie avec Visual Basic. Les contrôles ADO supportent des instructions de programmation supplémentaires (comme vous le verrez dans ce projet), mais, tel qu'il est décrit et utilisé ici, il est simple à comprendre.

But de l'application

La Figure PB9.1 montre la feuille que vous allez créer. Elle contient plusieurs lignes et contrôles ; sa création peut demander un certain temps. Elle propose un système de gestion de base de données complet de BIBLIO.MDB. Ce projet ne peut pas décrire toutes les actions du contrôle ADO, mais il décrit comment commencer à manipuler la base. Après avoir suivi ce projet, vous comprendrez une partie de ce qu'implique le travail avec les applications ADO.

Cette application imite quelque peu ce que peut produire l'assistant Création d'applications, mais dans ce projet, vous étudierez de l'intérieur les exigences des contrôles ADO. Vous apprendrez comment incorporer des instructions de programmation concernant les contrôles ADO qui permettent d'accéder aux tables de base de données sans intervention de l'utilisateur et de les modifier.

Figure PB9.1
Votre application ADO permettra de gérer cette base de donnés des livres.

Ce projet ne vous donnera pas une maîtrise complète des contrôles ADO et du langage de programmation sous-jacent. Vous en étudierez cependant les bases. Heureusement, de nombreux programmeurs Visual Basic n'auront jamais à programmer un contrôle ADO en utilisant en profondeur le langage dont nous allons parler. Ce projet a pour seul objectif de vous donner une introduction aux compétences nécessaires à une utilisation efficace du contrôle ADO.

Création de la feuille initiale

Pour débuter, créez la feuille initiale en plaçant les contrôles et en paramétrant leurs valeurs respectives comme il est décrit dans le Tableau PB9.1. Appuyez sur les touches Ctrl-T pour ouvrir la boîte de dialogue Composants et sélectionner Microsoft ADO Data Control 6.0 afin d'ajouter le contrôle à la Boîte à outils. Vous ajouterez d'autres propriétés à ces contrôles avant d'avoir achevé ce projet.

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille

Nom de propriété du contrôle	*Valeur de propriété*
Feuille : `Name`	frmBookTitle
Feuille : `Caption`	Application ADO — Titres des livres
Feuille : `Height`	4590
Feuille : `Width`	7740

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
ADO `Name`	adoBooks
ADO `Height`	735
ADO `Left`	5400
ADO `Top`	0
ADO `Width`	2055
Label #1 `Name`	lblApp
Label #1 `Alignment`	Center
Label #1 `BorderStyle`	Fixed Single
Label #1 `Caption`	Titre des livres
Label #1 `FontStyle`	Gras
Label #1 `FontSize`	18
Label #1 `Left`	2520
Label #1 `Height`	495
Label #1 `Top`	240
Label #1 `Width`	2650
Label #2 `Name`	lblTitle
Label #2 `Alignment`	Right Justify
Label #2 `Caption`	Titre:
Label #2 `FontSize`	10
Label #2 `Left`	720
Label #2 `Height`	255
Label #2 `Top`	840
Label #2 `Width`	510
Label #3 `Name`	lblYear
Label #3 `Alignment`	Right Justify
Label #3 `Caption`	Année de publication :

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
Label #3 `FontSize`	10
Label #3 `Left`	120
Label #3 `Height`	255
Label #3 `Top`	2400
Label #3 `Width`	1995
Label #4 `Name`	lblISBN
Label #4 `Alignment`	Right Justify
Label #4 `Caption`	ISBN :
Label #4 `FontSize`	10
Label #4 `Left`	3240
Label #4 `Height`	255
Label #4 `Top`	2400
Label #4 `Width`	615
Label #5 `Name`	lblPubID
Label #5 `Alignment`	Right Justify
Label #5 `Caption`	ID d'éditeur :
Label #5 `FontSize`	10
Label #5 `Left`	120
Label #5 `Height`	255
Label #5 `Top`	3000
Label #5 `Width`	1455
Label #6 `Name`	lblSubject
Label #6 `Alignment`	Right Justify
Label #6 `Caption`	Sujet :
Label #6 `FontSize`	10
Label #6 `Left`	3480

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
Label #6 `Height`	255
Label #6 `Top`	3000
Label #6 `Width`	855
TextBox #1 `Name`	txtTitle
TextBox #1 `DataField`	Title
TextBox #1 `DataSource`	adoBooks
TextBox #1 `Height`	1095
TextBox #1 `Left`	1320
TextBox #1 `Top`	840
TextBox #1 `Width`	5535
TextBox #2 `Name`	txtPub
TextBox #2 `DataField`	Year Published
TextBox #2 `DataSource`	adoBooks
TextBox #2 `Height`	345
TextBox #2 `Left`	2160
TextBox #2 `Top`	2400
TextBox #2 `Width`	975
TextBox #3 `Name`	txtISBN
TextBox #3 `DataField`	ISBN
TextBox #3 `DataSource`	adoBooks
TextBox #3 `Height`	345
TextBox #3 `Left`	3960
TextBox #3 `Top`	2400
TextBox #3 `Width`	3495
TextBox #4 `Name`	txtPubID
TextBox #4 `DataField`	PubID

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
TextBox #4 `DataSource`	adoBooks
TextBox #4 `Height`	345
TextBox #4 `Left`	1680
TextBox #4 `Top`	3000
TextBox #4 `Width`	1575
TextBox #5 `Name`	txtSubject
TextBox #5 `DataField`	Subject
TextBox #5 `DataSource`	adoBooks
TextBox #5 `Height`	345
TextBox #5 `Left`	4440
TextBox #5 `Top`	3000
TextBox #5 `Width`	1575
Bouton de Commande #1 `Name`	cmdSave
Bouton de commande #1 `Caption`	&Enregistrer
Bouton de commande #1 `Left`	240
Bouton de commande #1 `Top`	3600
Bouton de commande #1 `Width`	875
Bouton de commande #2 `Name`	cmdAdd
Bouton de commande #2 `Caption`	&Ajouter
Bouton de commande #2 `Left`	1200
Bouton de commande #2 `Top`	3600
Bouton de commande #2 `Width`	875
Bouton de commande #3 `Name`	cmdNew
Bouton de commande #3 `Caption`	&Nouveau
Bouton de commande #3 `Left`	2160
Bouton de commande #3 `Top`	3600

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
Bouton de commande #3 `Width`	875
Bouton de commande #4 `Name`	cmdDelete
Bouton de commande #4 `Caption`	&Supprimer
Bouton de commande #4 `Left`	3120
Bouton de commande #4 `Top`	3600
Bouton de commande #4 `Width`	875
Bouton de commande #5 `Name`	cmdCancel
Bouton de commande #5 `Caption`	Ann&uler
Bouton de commande #5 `Left`	4080
Bouton de commande #5 `Top`	3600
Bouton de commande #5 `Width`	875
Bouton de commande #6 `Name`	cmdPrev
Bouton de commande #6 `Caption`	&<
Bouton de commande #6 `Left`	5160
Bouton de commande #6 `Top`	3600
Bouton de commande #6 `Width`	495
Bouton de commande #7 `Name`	cmdNext
Bouton de commande #7 `Caption`	&>
Bouton de commande #7 `Left`	5760
Bouton de commande #7 `Top`	3600
Bouton de commande #7 `Width`	495
Bouton de commande #8 `Name`	cmdExit
Bouton de commande #8 `Caption`	&Quitter
Bouton de commande #8 `Left`	6600
Bouton de commande #8 `Top`	3600
Bouton de commande #8 `Width`	855

Tableau PB9.1 : Configurez ces contrôles et propriétés dans la feuille *(suite)*

Nom de propriété du contrôle	*Valeur de propriété*
Ligne #1 `Name`	Line1
Ligne #1 `X1`	120
Ligne #1 `X2`	7440
Ligne #1 `Y1`	2160
Ligne #1 `Y2`	2160
Ligne #2 `Name`	Line2
Ligne #2 `X1`	0
Ligne #2 `X2`	7800
Ligne #2 `Y1`	3480
Ligne #2 `Y2`	3480
Ligne #3 `Name`	Line3
Ligne #3 `X1`	5040
Ligne #3 `X2`	5040
Ligne #3 `Y1`	3480
Ligne #3 `Y2`	4280
Ligne #4 `Name`	Line4
Ligne #4 `X1`	6480
Ligne #4 `X2`	6480
Ligne #4 `Y1`	3480
Ligne #4 `Y2`	4280

En plaçant ces contrôles, notez que les zones de texte sont liées à leurs champs respectifs de la table BIBLIO.MDB, sur laquelle pointe le contrôle ADO `adoBooks`. Vous avez appris à réaliser le même type de liens avec le contrôle Data au Chapitre 18. Mais, contrairement à ce dernier, le contrôle ADO a plus de possibilités, comme vous le verrez au cours de ce projet.

Connexion du contrôle ADO aux données

Lorsque vous utilisez un contrôle ADO, vous devez le connecter aux données auxquelles il doit accéder, comme pour le contrôle Data ; le contrôle ADO peut cependant se connecter à des données qui ne sont pas nécessairement dans une table de base de données. Il permet aussi d'accéder à du courrier électronique, des graphiques et pratiquement toute source de données externe à l'application.

La plupart du temps, un programmeur se connectera à des bases de données, qu'elles soient locales ou en réseau. Votre première étape pour que le contrôle ADO fonctionne consiste donc à le connecter à sa source de données. Dans ce projet, cette source est la base de données Access 97 BIBLIO.MBD fournie avec Visual Basic.

Le contrôle ADO est bien plus difficile à connecter à une base de données que ne l'est le contrôle Data. Mais les avantages compensent largement cette complication due à une meilleure efficacité et à la possibilité de se connecter à quasi n'importe quel type de données.

Vous avez deux manières de vous connecter au contrôle ADO :

- configurer la propriété `ConnectionString`,
- utiliser du code pour connecter le contrôle aux données.

Si vous utilisez la fenêtre Propriétés pour vous connecter aux données, vous passez par des boîtes de dialogue qui simplifient le paramétrage de la connexion. L'autre méthode demande d'écrire du code assez abscons (cette section illustrera les deux manières).

La connexion par la fenêtre Propriétés n'est pas aussi simple que pour le contrôle Data. Lorsque vous cliquez sur la propriété `ConnectionString`, la boîte de dialogue illustrée à la Figure PB9.2 s'affiche. Vous devez déterminer les options qui correspondent à votre type de sources de données.

Dans cet exemple, comme pour la plupart des applications utilisant le contrôle ADO, vous créerez une simple chaîne de connexion qui pointe sur un fichier de base de données. Les deux premières options, Utiliser un fichier OLE DB (utilisé pour transmettre des données entre deux emplacements) et Utiliser un nom de source de données ODBC (utilisé avec les bases de données compatibles ODBC), ne sont pas nécessaires pour l'accès simple utilisé dans ce projet. Cliquez sur la troisième option pour spécifier une chaîne de connexion.

Figure PB9.2
La boîte de dialogue Pages de propriétés du contrôle ADO vous aide à spécifier une chaîne de connexion.

Les chaînes de connexion peuvent être assez longues, mais Visual Basic vous assiste dans leur création. Suivez ces étapes :

1. Cliquez sur le bouton Créer pour afficher la boîte de dialogue Propriétés des liaisons de données.
2. Double-cliquez sur la première option, Microsoft Jet 3.51 OLE DB Provider. (Les bases de données Access ont pour système sous-jacent la technologie Jet utilisée par Microsoft pour un accès rapide aux bases de données). L'onglet suivant, Connexion, s'affiche.
3. Cliquez sur les points de suspension à la droite de la première zone de texte et cherchez la base de données BIBLIO.MDB sur votre disque.
4. Cliquez sur le bouton Ouvrir pour lier la base de données au contrôle ADO.

Pour vous assurer que vous êtes correctement connecté à la base de données, cliquez sur le bouton Tester la connexion. Si les propriétés de la chaîne de connexion sont correctes, Visual Basic affiche une boîte de message qui vous signale un test réussi. Dans le cas contraire, vous pouvez corriger le problème.

5. Pour une vérification, cliquez sur l'onglet Tous pour voir la page résumant la chaîne de connexion, illustrée à la Figure PB9.3. Les autres éléments listés peuvent être modifiés si vous le souhaitez (vous pouvez par exemple paramétrer la sécurité d'accès).
6. Cliquez sur le bouton OK pour revenir à la boîte de dialogue Pages de propriétés, puis cliquez sur OK pour la fermer et revenir à la feuille.

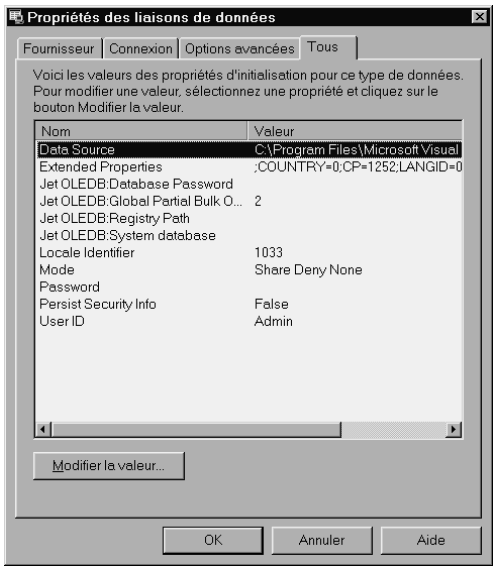


Figure PB9.3
Visual Basic récapitule les paramètres de votre chaîne de connexion.

Au lieu de sélectionner à l'aide de la boîte de dialogue, vous pouvez utiliser, n'importe où dans l'application, du code pour créer une chaîne de connexion. Par exemple, vous pouvez la créer au chargement de la feuille ou uniquement lorsque l'utilisateur a besoin d'accéder à la base. Plus vous attendez pour connecter les données, moins vous avez de risques qu'une coupure de courant ou un blocage de la machine affectent vos données ; l'application démarrera aussi plus rapidement.

Même pour utiliser du code pour accéder aux données, vous devez placer le contrôle ADO sur la feuille. Configurez sa propriété `Visible` *à* `False` *si vous ne voulez pas que l'utilisateur y accède autrement que par votre code. Vous ne pouvez pas utiliser à la fois le code et la fenêtre Propriétés pour configurer la chaîne de connexion, vérifiez donc que la propriété* `ConnectionString` *est bien vierge si vous utilisez du code dans votre programme pour configurer l'accès.*

Pour paramétrer le contrôle ADO par programme, vous devez placer l'instruction suivante dans la procédure événementielle `Form_Load()` :

```
adoBooks.ConnectionString = "Provider=Microsoft.Jet.OLEDB.3.51;"
➥& "Persist Security Info=False;" & "Data Source=C:\Program
➥Files\Microsoft Visual" & "Studio\VB98\Biblio.mdb"
```

L'instruction assigne exactement la même chaîne de connexion que celle créée au début de cette section à l'aide de la boîte de dialogue. Comme vous pouvez le voir, le format peut être assez épineux, mais la propriété `ConnectionString` comporte l'ensemble des informations de la boîte de dialogue.

Le contrôle n'est cependant pas encore tout à fait prêt à afficher les données de la base. Vous devez spécifier exactement quelles lignes de quelles tables sont à la disposition du reste de l'application. Vous devez utiliser SQL (un langage de programmation de base de données supporté par Visual Basic comme une propriété `RecordSource`) pour indiquer à Visual Basic d'accéder à tous les enregistrements (ce qu'indique le caractère générique `*`) de la table Titles de la base de données BIBLIO.MDB. L'instruction est la suivante :

```
adoBooks.RecordSource = "Select * From Titles"
```

SQL (Structured Query Language) signifie langage d'interrogation structuré. Il définit un langage de programmation universel (en théorie) pour l'accès aux bases de données. Vous pouvez émettre des commandes SQL à partir des applications Visual Basic.

Vous pouvez configurer la feuille à l'aide d'un contrôle ADO invisible pour accéder aux données en configurant la propriété `Visible` du contrôle ADO à `False`, effaçant toute valeur dans la propriété `ConnectionString` de la fenêtre Propriétés et assignant les propriétés `DataSource` et `DataField` de chaque contrôle de zone de texte telles qu'elles sont décrites dans le Tableau PB9.1. Dans ce cas, le code du programme doit permettre de parcourir les données selon les besoins. Dans cet exemple, les boutons de commande en bas de la feuille exigent du code pour permettre de parcourir les données. C'est l'objet de la section suivante.

Lorsque vous exécutez l'application avec les propriétés `ConnectionString` et `RecordSource` configurées par programme, tout en laissant les autres champs tels que vous les avez entrés à partir du Tableau PB9.1, aucune donnée de la base ne s'y affichera. Même si les champs de texte pointent tous sur le contrôle ADO, ce dernier n'est connecté à la base qu'au moment où il se charge et, à cet instant, les contrôles de texte ont échoué à trouver des données, car leur propriété `DataSource` (le contrôle ADO), n'est pas connectée à une base de données. Mais lorsque vous reliez le premier contrôle de texte à la source de données nouvellement connectée (comme le décrit la section suivante), tous les champs liés au contrôle ADO afficheront les données.

Code ou contrôle ?

Devez-vous laisser les utilisateurs commander le contrôle ADO à l'aide de ses boutons ou devez-vous le masquer et utiliser du code pour parcourir la table connectée ? Dans ce projet, vous étudiez les deux méthodes. Pour voir les données liées, un utilisateur peut cliquer sur les boutons du contrôle ADO ou sur les boutons en bas de la feuille.

Le choix de la méthode dépend de l'application, qui sert souvent à mettre à jour ou à afficher un enregistrement unique qui répond à certains critères. Elle doit alors pouvoir rechercher un enregistrement particulier ou parcourir l'ensemble des enregistrements pour calculer des totaux et des moyennes. Ces types de traitements sur une table exigent l'écriture de code, qui permet de parcourir les données en arrière-plan.

Recherche des données

Pour que le champ Title affiche les titres des livres, vous devez connecter sa propriété `DataSource` au contrôle ADO et sa propriété `DataField` au champ Title de la table. Pour réaliser cela à partir du code, vous pouvez utiliser les deux instructions suivantes :

```
Set txtTitle.DataSource = adoBooks
txtTitle.DataField = "Title"
```

Vous ne pouvez pas assigner ce champ dans la fenêtre Propriétés si vous connectez à l'exécution le contrôle ADO à la base par programme, comme l'explique la section précédente.

Une fois qu'une zone de texte est liée au contrôle ADO, vous pouvez répéter la même opération pour les autres ou utiliser les propriétés `DataSource` et `DataField` de la fenêtre Propriétés décrites au Tableau PB9.1, car elles fonctionneront une fois qu'un contrôle de la feuille est relié à la base de données nouvellement connectée.

Parcours des données

Lorsque vous exécutez l'application après avoir assigné la première zone de texte au contrôle ADO, les champs correspondants du premier enregistrement s'affichent comme l'illustre la Figure PB9.4

Les champs de la table ne contiennent pas tous des données précises et complètes. Par exemple, dans le premier enregistrement de la base de données BIBLIO.MDB, le champ Sujet est vide. Certains champs contiennent des valeurs étranges tel que `2nd` qui est sans doute l'édition, mais pas le sujet du livre..

Figure PB9.4
Le premier enregistrement peut maintenant s'afficher.

Seul le premier enregistrement s'affichera, car l'utilisateur n'a pas accès au contrôle ADO invisible et les boutons de commande ne sont pas encore connectés au code. Vous pouvez double-cliquer sur le bouton Suivant (le bouton dont le titre est >) pour ajouter son code d'activation. Le code qui suit déplace le pointeur sur l'enregistrement suivant de la table pour que le contrôle lié puisse accéder aux données :

```
1:  Private Sub cmdNext_Click()
2:     ' Avance d'un enregistrement dans les données
3:     If Not adoBooks.Recordset.EOF Then
4:        adoBooks.Recordset.MoveNext
5:        If adoBooks.Recordset.EOF Then
6:          adoBooks.Recordset.MovePrevious
7:        End If
8:     End If
9:  End Sub
```

Le jeu d'enregistrements utilisé dans les lignes 3 à 6 est la liste des enregistrements avec lesquels vous travaillez. Vous avez spécifié le jeu d'enregistrements précédemment par l'instruction SQL qui limitait les enregistrements à la table Titles. La méthode `MoveNext` du jeu d'enregistrement fait avancer le pointeur d'un enregistrement à la ligne 4. Tous les autres contrôles liés à la table avancent également.

Un problème peut se produire dans les situations suivantes, qu'il faut gérer :

- S'il n'y a aucun enregistrement dans la table de la base de données, une erreur se produira si vous tentez d'avancer.
- Si l'enregistrement en cours est le dernier de la table, une erreur se produira quand vous tenterez d'avancer.

L'instruction de la ligne 3 garantit qu'on n'a pas atteint la fin de la table en vérifiant la propriété `EOF` du jeu d'enregistrements, avec un opérateur `Not`. La ligne 3 signifie en

fait "si l'on n'est pas à la fin du jeu d'enregistrements, alors continuer". La ligne 5 garantit que, si la méthode `MoveNext` amène à la fin du fichier, la méthode `MovePrevious` fait alors reculer le pointeur pour l'empêcher de dépasser la fin de la table. `MovePrevious` est la méthode de jeu d'enregistrement qui permet de reculer dans la table d'un enregistrement.

Vous pouvez utiliser la méthode `MovePrevious` pour ajouter une procédure événementielle au bouton Précédent (dont le titre est <) :

```
1:  Private Sub cmdPrev_Click()
2:    ' Recule d'un enregistrement dans les données
3:    If Not adoBooks.Recordset.BOF Then
4:      adoBooks.Recordset.MovePrevious
5:      If adoBooks.Recordset.BOF Then
6:        adoBooks.Recordset.MoveNext
7:      End If
8:    End If
9:  End Sub
```

La propriété `BOF` du jeu d'enregistrements vérifie le début du fichier, pour garantir qu'on interdit à l'utilisateur de reculer lorsque le pointeur se trouve déjà au début.

Vous pouvez placer une instruction `End` dans la procédure événementielle `Click` du bouton de commande Quitter pour permettre à l'utilisateur de quitter le programme.

Mise à jour des tables

Le contrôle ADO met automatiquement à jour l'enregistrement de la table sous-jacente si l'utilisateur effectue une modification dans une zone de texte liée. Si vous exécutez l'application ultérieurement, les données modifiées s'affichent.

Cependant, vous pouvez également enregistrer des données dans la base de données en utilisant du code. L'écriture de ce type de code est longue et lourde et va bien au-delà de la portée de ce projet. Vous pouvez par exemple vouloir qu'un certain champ ne contienne que des valeurs uniques et émettre un message d'erreur si l'utilisateur entre une valeur incorrecte. Vous pouvez aussi récupérer des données provenant d'une autre source et les assigner dans la base de données.

Pour vous donner une idée de ce qu'implique une mise à jour d'une base de données à l'aide d'un contrôle ADO, la procédure d'enregistrement du Listing PB9.1 écrit les données des zones de texte dans la base de données. (Naturellement, c'est redondant dans le cadre de cette application, car les zones de texte sont déjà liées à la table).

Listing PB9.1 : Vous pouvez utiliser des méthodes d'écriture des données dans la table par programmation

```
 1:  Private Sub cmdSave_Click()
 2:    ' Assigne toutes les TextBox aux champs.
 3:    ' N'assigne que les données non nulles
 4:    ' (les lignes longues sont découpées)
 5:    adoBooks.Recordset!Title = _
 6:      IIf(txtTitle = "", "N/A", txtTitle)
 7:    adoBooks.Recordset![Year Published] = _
 8:      IIf(txtPub = "", "N/A", txtPub)
 9:    adoBooks.Recordset!ISBN = _
10:     IIf(txtISBN = "", "N/A", txtISBN)
11:   adoBooks.Recordset!PubID = _
12:     IIf(txtPubID = "", "N/A", txtPubID)
13:   adoBooks.Recordset!Subject = _
14:     IIf(txtSubject = "", "N/A", txtSubject)
15:
16:   ' Effectue la mise à jour réelle du recordset
17:   adoBooks.Recordset.Update
18:
19: End Sub
```

La fonction `IIf()` est utilisée tout au long du listing pour que la valeur `N/A` (*non applicable*) soit écrite quand un champ enregistré ne contient pas de données. `IIf()` garantit qu'aucune valeur écrite n'est nulle. Le nom de champ entre crochets de la ligne 7 est obligatoire, car il contient un espace. Visual Basic traite `[Year Published]` comme un nom de champ unique ; sans les crochets, il ne serait pas capable de reconnaître l'espace incorporé.

Le Listing PB9.1 n'ajoute que les données qui ont été modifiées dans la feuille du projet, mais vous pouvez facilement adapter le code pour rechercher dans une autre source les données à enregistrer dans la table. A la ligne 14, vous pourriez chercher des données externes, par exemple une saisie dans une zone de texte, au lieu d'aller dans la zone de texte de la feuille. La ligne 17 est nécessaire, car elle effectue la mise à jour effective dans la table, à l'aide de la méthode `Update`.

Le code du Listing PB9.1 écrit donc dans la table les modifications effectuées par l'utilisateur dans les données de la feuille (ces valeurs seront de toutes façons écrites, car les zones de texte sont liées à la base dans notre exemple). Pour ajouter de nouvelles données à la fin de la table, vous devez d'abord effacer les champs de la feuille pour permettre à l'utilisateur de saisir les données. (Vous pourriez connecter ce code au bouton de commande Nouveau, ce que cette application ne prend pas le temps de faire).

Avant de pouvoir écrire le nouvel enregistrement, il faut déplacer le pointeur de table à la fin du fichier pour qu'il accepte de nouvelles données (et non pas remplacer les données sur lesquelles il pointe) par ce code :

```
adoBooks.Recordset.AddNew   ' Préparer un nouvel enregistrement
```

Les données qui sont enregistrées ensuite apparaîtront comme un nouvel enregistrement dans la table.

Conclusion sur le contrôle ADO

Comme vous avez pu le constater, la programmation du contrôle ADO n'est pas aussi simple que le contrôle Data, mais sa puissance et sa rapidité sont à son avantage. Vous avez déjà passé pas mal de temps sur ce projet, et pourtant ses fonctionnalités — bien que proche d'une finalisation — sont encore incomplètes. Considérez les points suivants nécessaires à l'achèvement de la tâche :

- Lors de l'écriture du code des boutons de commande Ajouter ou Nouveau, vous devez effacer les champs de la feuille et mettre le curseur dans le premier champ pour que l'utilisateur puisse enregistrer l'enregistrement en cours ou en ajouter un nouveau.
- Les méthodes `AddNew` et `Update` mettent toujours la table à jour avec des données nouvelles ou modifiées de champs liés.
- Utilisez la méthode `MoveLast` pour déplacer le pointeur à la fin de la table avant d'ajouter un nouvel enregistrement.
- Programmez le bouton Annuler pour ne pas enregistrer les enregistrements nouveaux ou modifiés. La procédure événementielle du bouton de commande Annuler doit appeler une nouvelle fois la procédure d'affichage de la feuille pour remettre le code dans sa configuration antérieure.

Chapitre 19

Ajout d'un accès Internet

La leçon d'aujourd'hui montre comment accéder à l'Internet à partir des applications Visual Basic. Aucun didacticiel Visual Basic qui se respecte ne serait complet s'il ne mentionnait pas ses liens avec l'Internet, car c'est l'un des outils de programmation les plus simples actuellement disponibles. Gardez cependant à l'esprit le fait que même avec Visual Basic, la programmation des accès Internet reste une gageure. La leçon d'aujourd'hui n'en présente qu'un survol rapide.

Vous y étudierez :

- la connexion Internet de Visual Basic ;
- comment ajouter un navigateur Web à une application ;
- les contrôles Internet de Visual Basic, ;
- comment travailler avec l'encapsulation ;
- les documents ActiveX ;
- comment transformer pratiquement toute application Visual Basic en une application Internet.

L'assistant Internet

L'assistant Création d'applications effectue une partie du travail si vous souhaitez que votre application accède à l'Internet. Sélectionnez les choix adaptés et l'assistant ajoute l'accès à l'Internet à votre application, en lui donnant des capacités de communications planétaires.

Les outils Internet décrits dans cette leçon fonctionnent aussi bien sur l'Internet que sur un intranet. Les technologies Internet et intranet supportent un protocole commun, elles peuvent donc fonctionner avec le même type d'applications.

Un intranet est un système de réseau local (par exemple, un réseau dans un seul immeuble ou même une petite zone sur un seul étage) qui offre les mêmes fonctions que l'Internet.

Un protocole permet à deux ordinateurs de communiquer. Les connexions Internet et intranet utilisent un protocole commun : TCP/IP (Transfer Control Protocol/Internet Protocol). Il est utilisé pour les connexions Internet universelles.

Cette section explique le fonctionnement de l'assistant Création d'applications lorsqu'il est utilisé pour ajouter un accès à l'Internet à une application. Si l'assistant supporte l'accès, il donne plus particulièrement à votre application la capacité de parcourir les pages du World Wide Web.

Les utilisateurs de votre application doivent déjà avoir un fournisseur de services Internet pour pouvoir accéder au Web à l'aide de votre programme. Vous devez également disposer d'Internet Explorer 4 ou d'une version postérieure sur votre système de développement pour pouvoir travailler sur le support complet d'Internet de Visual Basic. Lors de son installation, ce dernier propose en option d'installer Internet Explorer 4.

Un fournisseur de services Internet est une entreprise qui propose des comptes de connexion à l'Internet. Les sociétés qui disposent d'une connexion directe à l'Internet n'ont pas besoin de fournisseur de services.

Lorsque vous créez le noyau d'une application avec l'assistant Création d'applications, la sixième boîte de dialogue (voir Figure 19.1) est la fenêtre Connexion à Internet, qui configure l'accès au Web de l'application que vous créez.

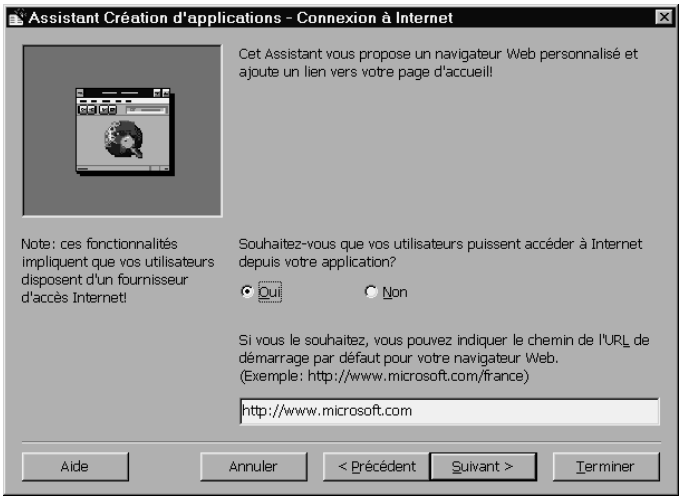


Figure 19.1
Vous pouvez sélectionner l'accès à l'Internet dans la boîte de dialogue Connexion à Internet de l'assistant.

L'assistant propose un URL par défaut : la page d'accueil de Microsoft. Vous devez modifier cet URL pour présenter autre chose aux utilisateurs. Lorsque le navigateur est déclenché dans l'application, il se connecte (en utilisant le fournisseur de services Internet existant) à l'URL spécifié dans l'assistant. Vous pouvez, par exemple, inclure l'URL de la page d'accueil de votre entreprise dans la zone de texte. Lorsque vous sélectionnez l'option Oui, l'assistant insère en fait un moteur de navigateur Web dans l'application générée.

Un URL (Uniform Ressource Locator ou Localisateur uniforme de ressource) est une adresse de site Web. Chaque site dispose d'une adresse URL unique.

Faites toujours commencer un URL par `http://`. (*http* signifie *Hypertext Transfer Protocol*, et désigne la procédure de communication standard utilisée pour l'accès aux pages Web). La majorité des navigateurs modernes se passent du préfixe `http://`, mais l'assistant l'exige.

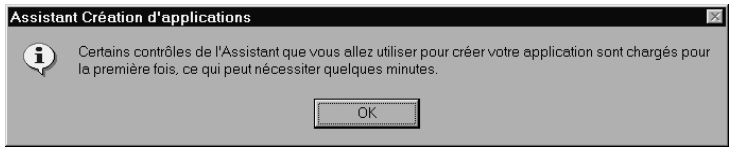
Vous pouvez rapidement créer une application de test pour accéder à l'Internet. Suivez ces étapes pour ajouter une fonction de navigation Internet dans l'application créée par l'assistant :

1. Créez un nouveau projet et lancez l'assistant Création d'applications.
2. Cliquez sur Suivant pour sauter la première boîte de dialogue.
3. Sélectionnez Interface monodocument (SDI) pour simplifier l'application générée.
4. Cliquez sur Suivant dans les quatre boîtes de dialogue suivantes pour accepter les options de menu par défaut.

5. Dans la boîte de dialogue Connexion à Internet, cliquez sur Oui. Laissez, dans cet exemple, l'adresse URL du site Web de Microsoft.
6. Cliquez sur Terminer pour générer l'application.

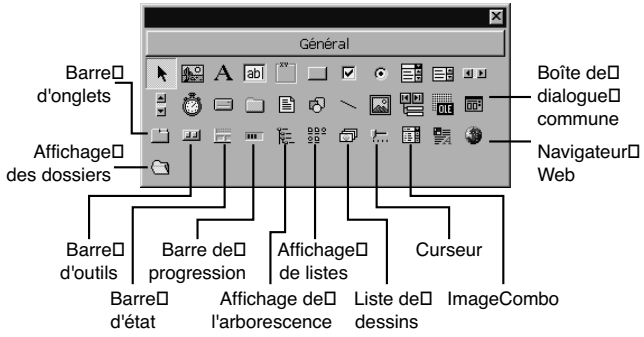
Si c'est la première fois que vous utilisez les contrôles Internet depuis l'installation de l'environnement Visual Basic, vous verrez la boîte de dialogue de la Figure 19.2. Cliquez sur OK pour préparer ces contrôles et les charger dans l'environnement Visual Basic.

Figure 19.2
Visual Basic doit préparer les contrôles Internet pour leur première utilisation.



Lorsque vous revenez dans l'environnement Visual Basic, observez la fenêtre Boîte à outils. La Figure 19.3 montre les outils ajoutés par l'assistant à la collection des contrôles internes. Vous en avez déjà utilisé certains au cours de ce didacticiel : la boîte de dialogue standard, la barre d'outils, la liste de dessins et le curseur.

Figure 19.3
L'assistant a ajouté de nouveaux outils à la boîte à outils.



Les outils supplémentaires donnent aux parties de navigation Web de l'application les contrôles dont elles ont besoin. A l'évidence, le contrôle Navigateur Web est l'outil principal concerné par cette leçon.

Appuyez sur F5 pour exécuter ce noyau d'application. L'écran que vous voyez n'est pas différent des autres écrans d'applications générées par des assistants que vous avez déjà pu rencontrer. Le menu et la barre d'outils par défaut s'affichent dans la feuille. La fonc-

tion Internet apparaît cependant lorsque vous choisissez Navigateur Web dans le menu Affichage. Une boîte de dialogue de navigation Web s'affiche et lance la connexion à l'aide de la fenêtre classique de connexion à votre fournisseur Internet (à moins que vous ne soyez déjà connecté ou que vous ne disposiez d'une capacité de connexion automatique). Après avoir saisi le nom d'utilisateur et le mot de passe, une fenêtre de type Internet Explorer s'affiche au centre de la feuille de l'application et présente le site Web de Microsoft (voir Figure 19.4).

Figure 19.4
L'application générée par l'assistant est connectée au site Web de Microsoft.

La fenêtre Internet Explorer que vous voyez est en fait une petite application qui enrobe un énorme contrôle ActiveX. Le navigateur Web inséré par l'assistant de Visual Basic est un exemple de contrôle ActiveX. S'il est plus simple que la version complète d'Internet Explorer (il y a moins de boutons de barre d'outils et pas de menu), il fournit toutes les fonctions de navigation nécessaires : les boutons Précédent et Suivant, Origine, etc. Si vous cliquez sur Rechercher, Internet Explorer utilise le site de recherche de Microsoft pour lancer la requête.

Pour vous déconnecter de l'Internet, vous devez fermer le navigateur Web, double-cliquer sur l'icône de connexion de votre fournisseur Internet dans la Barre des tâches, et sélectionner l'option Déconnecter. Le navigateur ne dispose pas de fonction de déconnexion, mais vous pouvez l'ajouter par programmation.

Etude de quelques contrôles Internet

Si vous utilisez Visual Basic dans ses éditions Professionnelle ou Entreprise, vous pouvez utiliser plusieurs contrôles ActiveX Internet avancés pour ajouter et commander l'accès à l'Internet à l'intérieur de votre application. L'exemple de la section précédente a montré la puissance d'un seul contrôle : le navigateur Web.

Plusieurs contrôles Internet s'affichent lorsque vous choisissez Composants dans le menu Projet. Cette section les étudie et explique quand et comment les utiliser dans des projets qui accèdent à l'Internet.

L'accès Internet peut avoir plusieurs significations dans le monde actuel — par exemple, il peut faire référence à une application complète à laquelle l'utilisateur accède et qu'il exécute sur le Web. L'Internet offre de nos jours bien d'autres services que l'affichage des pages Web et le téléchargement de fichiers, en particulier avec tous les nouveaux contrôles ActiveX disponibles, qui fonctionnent au travers de l'Internet aussi aisément qu'ils le font dans les applications situées sur un ordinateur unique. Lorsque vous activez des pages Web avec des programmes, Visual Basic peut être leur moteur.

Les contrôles d'encapsulation

Le terme *encapsulation* fait référence à différentes choses, suivant que vous encapsulez des données, du code ou les deux. Mais, au sens large, il signifie toujours *empaquetage*. Visual Basic comprend certains contrôles Internet qui encapsulent, ou empaquettent, les applications existantes pour qu'elles fonctionnent avec la technologie Internet.

L'encapsulation se rapporte à l'empaquetage des composants, tel qu'il se produit avec les objets Visual Basic qui supportent des propriétés, des méthodes et des événements.

Voici une liste des contrôles d'encapsulation :

- **Contrôle de transfert Internet.** Il encapsule les trois protocoles Internet les plus courants : HTTP, FTP (*File Transfer Protocol*) et Gopher (un protocole de recherche permettant de trouver des informations sur l'Internet). Vous pouvez télécharger des fichiers depuis vos applications Visual Basic en utilisant FTP.
- **Contrôle navigateur Web.** Il encapsule un navigateur Web dans votre application.
- **Contrôle Winsock.** C'est un contrôle Windows commun de connexion et d'échange de données qui propose deux protocoles : UDP (*User Datagram Protocol*) et TCP (*Transmission Control Protocol*).

Vous avez déjà vu l'un de ces contrôles — le navigateur Web — à la section précédente. L'assistant Création d'applications l'utilise pour insérer un navigateur dans l'application générée. Comme vous avez pu le constater, il n'est pas aussi complet qu'Internet Explorer, mais il fournit simplement un accès direct à l'Internet à tout utilisateur qui est inscrit à un service Internet.

Contrôle Internet Explorer

Visual Basic est livré avec plusieurs contrôles que vous pouvez ajouter à vos projets pour qu'ils interagissent avec le Web à l'aide de la technologie d'Internet Explorer. Ils commencent tous par les lettres *IE* dans la boîte de dialogue Composants.

Le Tableau 19.1 vous aide à localiser les contrôles décrits dans cette section, car leur nom n'est souvent pas suffisant pour décrire leurs caractéristiques. (Pour accéder à la boîte de dialogue Composants, sélectionnez Composants dans le menu Projet.)

Tableau 19.1 : Plusieurs composants comprennent des caractéristiques apparentées à Internet Explorer, comme d'autres contrôles Internet

Nom de composant	*Description*
IE Animated button	Affichage animé montrant la connexion d'Internet Explorer
IE Popup Menu	Contrôle de menu contextuel sur la page Web
IE Preloader	Précharge une page d'un site avant que l'accès visible à l'Internet commence
IE Super Label	Etiquette de page Web
IE Timer	Propose des opérations de timing pour les services Internet
Microsoft Internet Controls	Contrôle de navigateur Web
Microsoft Internet Transfer Control 6.0	Contrôle TCP
Microsoft Winsock Control 6.0	Connexion Windows à des protocoles Internet courants

Si vous utilisez le service en ligne Microsoft Network, un ensemble de contrôles est livré avec Visual Basic permettant aux applications que vous écrivez de bénéficier de services liés à Microsoft Network, comme le contrôle de courrier électronique MSN. Ces contrôles commencent par les lettres MSN dans la boîte de dialogue Composants.

Présentation rapide de points avancés

Vous avez déjà eu un aperçu de ce qui est disponible afin d'utiliser Visual Basic pour interagir avec l'Internet. La méthode la plus simple pour ajouter des caractéristiques Internet à vos applications est d'utiliser l'assistant Création d'applications, comme vous l'avez déjà fait dans cette leçon. Pour aller plus loin, vous avez devant vous un chemin d'apprentissage assez escarpé.

Cette section étudie quelques termes et concepts auxquels vous aurez à faire face lorsque vous commencerez à creuser les relations de Visual Basic à l'Internet. En apprenant ce qui est disponible actuellement, vous aurez quelques bases le jour où vous apprendrez les détails nécessaires pour fournir une interaction complète à l'Internet depuis vos applications.

Documents ActiveX

Afin de développer une application Visual Basic exclusivement Internet, vous pouvez utiliser les documents ActiveX pour commencer. Un *document ActiveX* ressemble à et fonctionne comme une application Visual Basic dans une fenêtre feuille, mais il envoie vers l'ordinateur de l'utilisateur final tous les contrôles ActiveX utilisés, s'ils y sont absents. Le document se présente à l'utilisateur comme une page Web HTML classique. Le document ActiveX peut, par exemple, contenir des liens hypertexte (des contrôles ActiveX qui sont téléchargés ou utilisés, suivant le contenu de la machine de l'utilisateur). Les menus des documents ActiveX peuvent aussi être fusionnés automatiquement avec l'application parente (comme les serveurs OLE).

HTML (Hypertext Markup Language) est le principal langage de mise en pages du Web. Un navigateur Web peut déchiffrer tous les codes HTML, formater et afficher la page Web et l'application décrites par le code HTML. La section finale de la leçon d'aujourd'hui montre un exemple de code HTML. Un fichier HTML est un fichier texte pur qui contient des codes HTML. Si vous pouvez créer un fichier HTML en utilisant un simple éditeur de texte, il existe de nombreux outils visuels permettant de concevoir et de créer des pages Web sans avoir à taper le moindre code HTML. Même

Visual Basic peut vous aider dans la conception de pages Web si vous sélectionnez l'assistant DHTML. Les fichiers HTML ont l'extension .HTM extension de fichier.

Les éditions Professionnelle et Entreprise de Visual Basic 6 supportent *DHTML* (*Dynamic Hypertext Markup Language*) qui réagit à des actions sur la page Web. Le document ActiveX est lié à une page HTML que vous créez ou utilisez. Quand l'utilisateur clique sur le lien vers votre document ActiveX, ce dernier s'active, le contrôle arrive sur l'ordinateur de l'utilisateur, puis le code du document ActiveX de la page Web s'exécute quand la page s'affiche. Le contenu est dynamique, ce qui signifie que des paramètres tels que la couleur et le style du texte peuvent différer d'une machine à l'autre en fonction des configurations.

Un document ActiveX n'est pas statique. C'est en fait une véritable application qui s'exécute. L'utilisation d'un concept de document aide les programmeurs à voir comment les pages Web utilisent le document ActiveX incorporé.

La raison principale pour créer des documents ActiveX est sans doute qu'Internet Explorer peut les exécuter comme s'il était un programme de contrôle ou un lanceur de programme du système d'exploitation. Les menus du document ActiveX fusionnent avec les menus d'Internet Explorer (et en remplacent les fonctionnalités si nécessaire), et vous n'avez pas à apprendre un nouveau langage, tel que Java, pour activer les pages Web.

Java est un langage de programmation de l'Internet, fondé sur C++, utilisé pour activer les pages Web et interagir avec les utilisateurs en transmettant de petits programmes Java, nommés applets, avec la page Web. Les applets s'exécutent sur la machine de l'utilisateur lorsqu'il consulte une page Web qui les contient.

La boîte de dialogue Nouveau projet contient deux icônes — Exe Document ActiveX et Dll Document ActiveX — qui créent des noyaux de documents ActiveX. La Figure 19.5 montre ces icônes. Vous pouvez, à partir de là, ajouter toutes les caractéristiques voulues à la feuille, comme vous le faites avec les applications classiques.

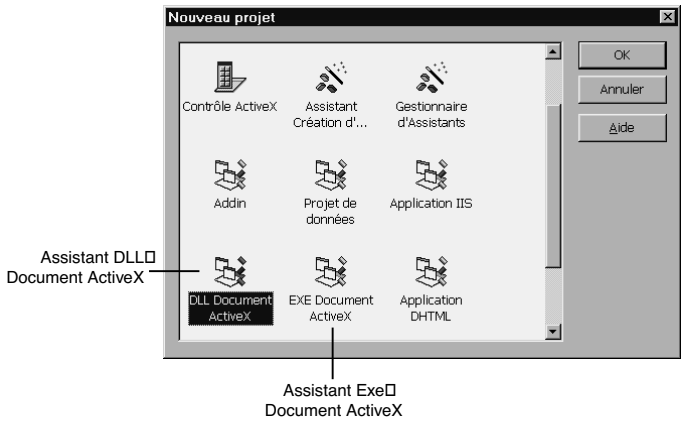


Figure 19.5
Démarrez l'assistant Nouveau projet pour voir les assistants Documents ActiveX.

L'assistant Migration de document ActiveX

Le plus simple pour porter vos applications vers l'Internet est de laisser Visual Basic le faire à votre place. Lorsque vous ajoutez l'assistant Migration de document ActiveX en choix du menu Compléments, vous ajoutez un outil puissant qui peut transformer presque toute application Visual Basic en une application Internet. Vous pouvez placer vos applications, converties en documents ActiveX, sur un serveur Web ; les utilisateurs pourront alors interagir avec elles au travers d'un navigateur Web, comme avec les pages Web qui contiennent du code HTML, VBScript et Java.

Pour voir la simplicité de la conversion d'une application en application compatible Internet, suivez ces étapes pour utiliser l'assistant Migration de documents ActiveX :

1. Sélectionnez Compléments, Gestionnaire de compléments et double-cliquez sur l'entrée Assistant Migration de document ActiveX VB6 pour ajouter l'assistant à la liste.
2. Cliquez sur OK pour fermer la boîte de dialogue Gestionnaire de compléments.
3. Ouvrez l'application exemple Calc.Vbp, fournie avec Visual Basic. Cherchez dans le dossier Samples du CD-ROM ou sur le disque dur si vous avez installé les exemples lors de l'installation de Visual Basic.
4. Appuyez sur F5 pour exécuter l'application. Comme le montre la Figure 19.6, elle simule une calculette de poche.
5. Arrêtez l'exécution de l'application.
6. Sélectionnez Compléments, Assistant Migration de document ActiveX.

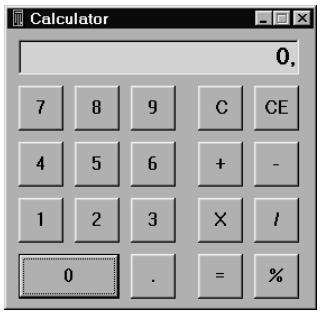


Figure 19.6
L'application Calc imite les fonctions d'une calculette de poche.

7. Cliquez sur Suivant pour sauter la fenêtre d'introduction.
8. La seconde fenêtre s'affiche (voir Figure 19.7) ; elle liste toutes les feuilles du projet en cours. Le projet Calc.vbp ne contient qu'une feuille, cliquez sur l'entrée Calculator pour la sélectionner.

Si votre projet comprend plusieurs feuilles, vous pouvez choisir de n'en transmettre que quelques-unes dans des documents ActiveX compatibles Web. Chaque feuille devient un document ActiveX indépendant.

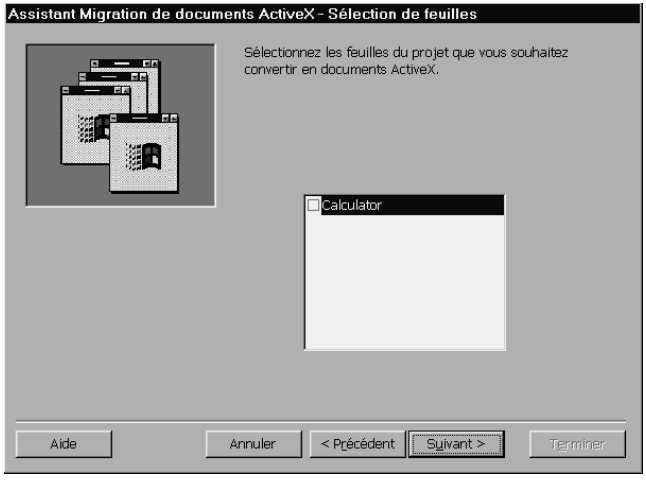


Figure 19.7
Sélectionnez la feuille de l'application qui apparaîtra sur la page Web comme un document ActiveX.

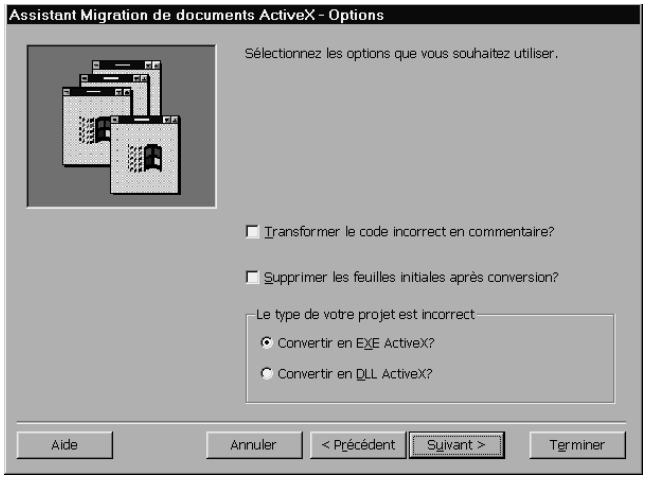
9. La fenêtre Options (voir la Figure 19.8) détermine comment l'assistant doit gérer les éléments qu'il ne peut pas convertir. Certains types d'instructions de communication avancées ne peuvent pas fonctionner dans un document ActiveX, même si la majeure

partie du code Visual Basic fonctionnera en douceur. Si vous cochez la première option, Visual Basic placera des commentaires devant toutes les instructions que l'assistant ne peut pas convertir. L'application résultante sera incomplète, mais vous pourrez rechercher les commentaires et supprimer le code ou le corriger s'il est vital de convertir l'application. Il n'y a pas de code incorrect dans l'application Calc.vbp.

En outre, vous pouvez choisir de supprimer du projet les feuilles converties, car elles se retrouveront dans un document ActiveX une fois l'assistant achevé. Laissez cochée l'option Convertir en EXE ActiveX pour que l'assistant crée un exécutable, et pas une DLL.

Une DLL (Dynamic Link Library), ou bibliothèque de liens dynamiques est une routine compilée qui peut être partagée par une ou plusieurs applications compilées.

Figure 19.8
Configurez les options du document ActiveX à afficher.



Dans le cadre de cet exemple, laissez les paramètres par défaut de la boîte de dialogue Options et cliquez sur Suivant pour continuer.

10. Laissez les options telles quelles et cliquez sur Terminer dans la fenêtre Terminé! pour démarrer la migration de l'application Calc.vbp vers un document ActiveX. Une fois qu'elle s'achève, une boîte de dialogue de fin s'affiche.
11. Cliquez sur OK pour fermer la boîte de dialogue de fin.

Après chaque Migration, une fenêtre de récapitulatif s'affiche(voir Figure 19.9). La fenêtre Récapitulatif est importante, car les instructions qu'elle présente vous permettent de savoir ce qu'il reste à faire pour tester la migration. Après avoir lu le rapport, vous pouvez cliquer sur le bouton Enregistrer pour enregistrer le texte et fermer la fenêtre. Dans cet exemple, vous pouvez fermer la fenêtre sans l'enregistrer.

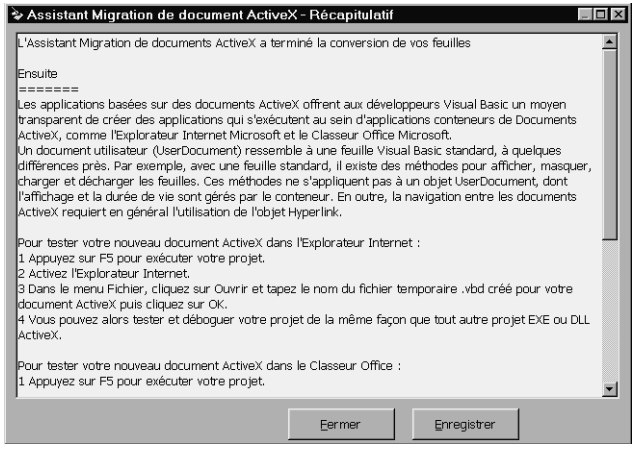


Figure 19.9
La fenêtre Récapitulatif décrit les actions à accomplir pour achever la migration.

La migration étant achevée, vous devez exécuter encore une fois l'application dans l'environnement Visual Basic pour préparer un objet document ActiveX exécutable. D'une certaine manière, Visual Basic compile votre application, mais contrairement à une compilation classique, l'exécution de l'application convertie crée un document ActiveX avec une extension .VBD (*Visual Basic Document*). Vous devrez encore compiler le document ActiveX dans un format EXE pour l'utiliser hors de l'environnement Visual Basic.

Dès que vous démarrez l'application, Visual Basic affiche la boîte de dialogue Propriétés du projet (voir Figure 19.10). Elle détermine le comportement du document lorsque vous exécuterez le programme. Si vous observez la fenêtre Projet, vous constaterez qu'elle comprend deux composants : la feuille `Calc` habituelle et un second objet, `docCalculator`. Ce dernier est le composant document ActiveX créé quand vous avez appuyé sur F5 pour exécuter l'application.

Lorsque vous cliquez sur OK, Visual Basic lance le document ActiveX dans le navigateur Internet. La Figure 19.11 montre le résultat. Remarquez ce qui s'est produit :

- Le navigateur Internet exécute en fait votre application Visual Basic.
- Vous n'avez pas écrit une ligne d'instructions HTML pour générer la page Web.

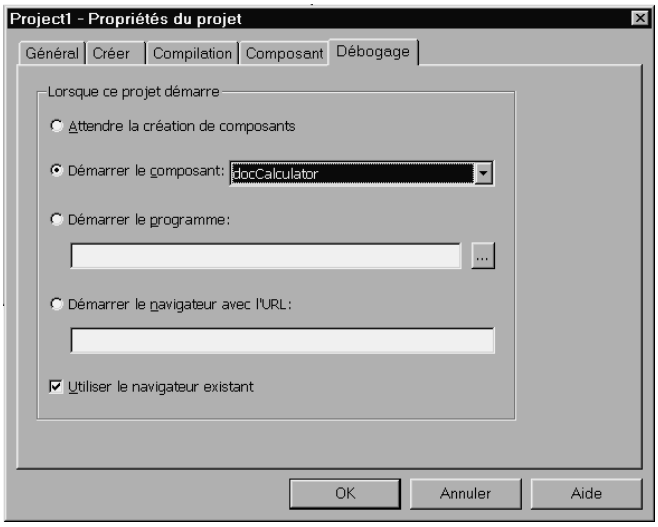


Figure 19.10
Spécifiez les propriétés du projet de document ActiveX.

- L'application Calculette est transformée avec succès en une application Web.
- Si vous fermez le projet, vous pouvez à tout moment redémarrer le navigateur Internet, sélectionner Fichier, Ouvrir, rechercher le fichier docCalculator.vbd et exécuter l'application compilée sous forme de document ActiveX, sans démarrer Visual Basic.
- Si vous placez l'application sur votre serveur Web (en considérant que vous avez accès à un serveur Web où vous pouvez proposer des pages Web sur l'Internet), tout utilisateur d'Internet dans le monde entier qui utilise un navigateur compatible ActiveX sera capable d'exécuter cette application. (Cela simule en fait l'action d'une applet Java).

Une fois le navigateur Internet fermé, vous pouvez sélectionner Exécution, Fin pour arrêter l'exécution de Visual Basic.

Tout conteneur de contrôle ActiveX peut contenir un document ActiveX. Il est important de le comprendre, car cela montre l'universalité des documents ActiveX. En d'autres termes, si vous utilisez une application, par exemple, une application de dessin ou un autre langage de programmation, qui accepte l'incorporation de contrôles ActiveX, vous pourrez ajouter vos documents ActiveX à cette application. Document ActiveX n'est qu'une manière savante de désigner un contrôle ActiveX migré depuis une application Visual Basic. Le second exercice montre comment utiliser cette fonction si Microsoft Office est installé sur votre machine.

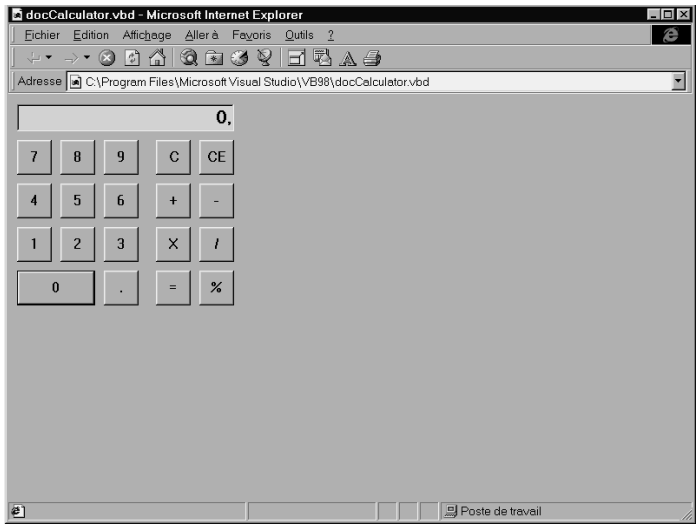


Figure 19.11
Internet Explorer est la plate-forme sur laquelle l'application Calculette s'exécute désormais.

HTML et VBScript

S'il suffit de connaître le langage de programmation Visual Basic pour accéder aux fonctions de connexion Internet de Visual Basic 6, vous devez maîtriser deux autres langages pour bien lier le tout. HTML est le langage de mise en forme des pages Web ; il est conçu pour atteindre les objectifs suivants :

- formater les pages Web en colonnes, avec des graphiques et des titres adaptés ;
- permettre l'intégration de programmes de service supplémentaires de l'Internet, comme les documents ActiveX et Java (un petit langage de programmation pour rendre actives les pages Web).

HTML est désigné comme un langage de *script*. Il ne se compile pas pour donner un exécutable comme le font les programmes Visual Basic. HTML formate les pages Web, spécifie où mettre les graphiques et les cadres de séparation, et permet d'activer des applications incorporées comme les documents ActiveX et les programmes Java.

VBScript, comme son nom l'indique, est un autre langage de script, conçu par Microsoft à partir du langage de programmation Visual Basic. Il est utile lorsque vous voulez ajouter des caractéristiques clés de Visual Basic à une page Web, comme un message surgissant, une boîte de saisie, un calcul en boucle, etc. VBScript, malgré ses bases Visual Basic, ne remplace pas les documents ActiveX, mais il permet de les charger dans une page HTML pour qu'ils s'exécutent. C'est donc le support permettant aux documents HTML de trouver et d'exécuter des documents ActiveX Visual Basic.

VBScript n'a pas été conçu à l'origine pour être exclusivement un lanceur de documents ActiveX — en fait, VBScript existait avant ActiveX. Le chargement des documents ActiveX dans les pages HTML n'est qu'une des nombreuses tâches dont est capable VBScript, mais sans doute la plus importante pour un programmeur VB6.

Le Listing 19.1 montre un exemple des premières lignes du code HTML du site Web Microsoft.

Listing 19.1 : Quelques lignes de code HTML peuvent révéler comment fonctionne le code de formatage des pages Web

```
<HTML>
<HEAD>
<TITLE>MSN.COM</TITLE>
<meta http-equiv="Content-Type" content="text/html;
  charset=iso-8859-1">
<META http-equiv="PICS-Label" content=
 '(PICS-1.0 "http://www.rsac.org/ratingsv01.html"
 l comment "RSACi North America Server" by
 "Microsoft Network"'>
</HEAD>
<FRAMESET rows="20,*" frameborder="0"
 framespacing="0" border="0">
<FRAME src="/pilot.htm" name="pilot"
 NORESIZE scrolling="no" marginwidth="0"
 marginheight="0" frameborder="0" framespacing="0">
</FRAMESET>
</html>
```

Connectez-vous à l'Internet et allez avec votre navigateur Web sur la page d'accueil de Microsoft à **http://www.microsoft.com/**. Même si la page peut différer de celle décrite dans le Listing 19.1, ce n'est pas ce texte que vous verrez. HTML est un langage de description de mise en forme de page. Les commandes du Listing 19.1 indiquent à votre navigateur Web comment afficher le texte et les graphiques qui sont transmis vers votre machine lorsque votre navigateur pointe la page.

Le Listing 19.2 montre un extrait d'un exemple VBScript. Vous pouvez en comprendre une bonne partie grâce à votre connaissance du langage Visual Basic.

Listing 19.2 : Un exemple de VBScript qui montre les ressemblances avec Visual Basic

```
<SCRIPT Language="VBScript">
        Call PrintWelcome
        Call ModMessage
```

```
Sub PrintWelcome
                If Date() = "2/2/98" Then
                       document.write ". . . .Anniversaire de Cathy !"
                End If
                If Date() = "2/5/98" Then
                       document.write ". . . .Anniversaire d'Eric !"
                End If
                If Date() = "5/17/98" Then
                       document.write ". . . .Anniversaire de Michael !"
                End If
                If Date() = "7/25/98" Then
                       document.write ". . . .Mon Anniversaire !"
                End If
        End Sub
Sub ModMessage
        Document.Write "<BR>Dernière modification de cette page :
        ➥"+Document.lastModified +"</FONT><BR>"
End Sub
</SCRIPT>
```

De VB à Java ?

Une technique que vous pouvez vite étudier est la conversion de Visual Basic vers Java. Certains fournisseurs vendent déjà de tels outils et d'autres ont annoncé leur intention de le faire. Le gros avantage de ce type d'outil est que vous n'avez pas à trop vous préoccuper des contrôles Internet. Si vous pouvez écrire une application qui utilise des contrôles Visual Basic, le programme de conversion traduit le projet Visual Basic en projet Java. Dans Java, vous pouvez incorporer l'application dans vos pages Web intranet ou Internet, et l'application aboutit sur l'écran de l'utilisateur dès qu'il affiche la page Web.

Ces outils de conversion Java ne remplacent pas nécessairement l'assistant Migration de document ActiveX dont nous avons déjà parlé. Cependant, sur certains systèmes autres que Windows supportant Java, mais pas ActiveX, les applications Java peuvent être plus universellement acceptées que les applications basées sur ActiveX.

Si Java est une nouveauté pour vous, disons qu'il fournit un véritable contenu actif aux pages Web depuis bien avant l'apparition des contrôles ActiveX. Java fonctionne au travers des pages Web et il s'exécute sur la machine de l'utilisateur final, même si cette machine et le système d'exploitation ne sont pas les mêmes que ceux du développeur. Pour des informations supplémentaires sur Java, reportez-vous à l'abondante littérature existant sur le sujet.

Visual J++ est l'implémentation Microsoft de type Java. Visual J++ comprend une interface programmatique qui ressemble à l'interface de Visual Basic et fonctionne de manière très proche. Les deux environnements supportent le style Visual Studio. Vous serez donc déjà familiarisé avec l'environnement de programmation si vous utilisez Visual J++ comme générateur de langage Java.

Types d'applications Internet Visual Basic

Visual Basic peut créer les deux types d'application Internet suivantes :

- **Application IIS.** La manière la plus simple d'incorporer Internet à des applications Visual Basic est de le faire sous la forme d'applications IIS (*Internet Information Server*). Le navigateur Web incorporé à l'application créée plus tôt avec l'assistant Création d'application est une application IIS. Le serveur gère tout le traitement des commandes Visual Basic.
- **Applications DHTML.** Permet d'écrire du code qui réagit à des événements sur une page Web HTML. Le navigateur de l'utilisateur final interprète et exécute ces commandes, et le serveur distant n'a pas grand-chose d'autre à faire que de répondre aux requêtes particulières quand elles se produisent, comme, par exemple, récupérer des pages Web supplémentaires.

N'oubliez pas que la leçon d'aujourd'hui ne propose qu'un survol de la création d'applications Visual Basic. Ne vous attendez pas à maîtriser des concepts Internet avancés en un jour. Peu importe votre niveau en programmation, entrer dans le monde d'un programmeur Internet, un peu comme pour un programmeur de base de données, exige une formation sur la gestion des protocoles de connexion, les langages de script, les contrôles ActiveX traitant de la communication, et la programmation client/serveur.

Dans la terminologie en ligne, le client est l'application qui accède à l'Internet, et le serveur est l'ordinateur qui détient les pages Web que l'utilisateur final (le client) affiche et sur lesquelles il interagit. Le monde en ligne est basé sur les transactions ; c'est-à-dire que l'utilisateur émet une requête, par exemple, dans le navigateur Web et le serveur traite cette requête, puis envoie une transaction résultante à l'utilisateur sous la forme d'une page Web ou d'une applet ActiveX.

Visual Basic supporte ces deux types de développement d'applications Internet ; vous pouvez donc les développer dans Visual Basic, et les tester avec les outils de débogage que vous apprendrez dans le Chapitre 21.

En résumé

Ce chapitre a donné une brève présentation du rôle de Visual Basic en tant qu'acteur Internet. A l'évidence, cette leçon ne peut couvrir qu'une très faible partie des détails nécessaires pour vraiment transformer Visual Basic en un outil de programmation Internet. Des bases solides sont nécessaires, ne serait ce que dans la technologie Internet, avant de s'attaquer à l'interface de Visual Basic. Il existe quantité de bons livres et de références en ligne, mais la meilleure chose est déjà d'étudier le matériel qui accompagne Visual Basic 6. Vous y trouverez des descriptions qui détaillent votre rôle en tant que programmeur Internet.

Ne soyez pas effrayés par l'idée d'écrire des applications qui interagissent avec l'Internet. Rendez-vous compte que l'objectif annoncé de faire tenir chaque leçon sur la durée d'une journée raisonnable n'aurait pas pu être atteint si la leçon d'aujourd'hui s'était aventurée à vous apprendre les nombreuses particularités nécessaires à l'écriture des programmes Internet. En tout cas, les programmeurs Internet sont bien récompensés pour leurs compétences résultant des études approfondies exigées et du rythme qu'ils doivent suivre pour rester à jour dans la technologie.

La leçon de demain décrit comment créer des pages d'aide pour les applications Visual Basic. Vous verrez de plus près comment fonctionne HTML en créant un système d'aide HTML et en utilisant des pages de type HTML.

Questions-réponses

Q Mon application comprend trois feuilles. L'assistant Migration de document ActiveX compile-t-il ces trois feuilles en un document ActiveX unique ?

R L'assistant Migration de document ActiveX convertit les feuilles, non les applications entières, en documents ActiveX. En d'autres termes, si votre application contient quatre feuilles, chacune générera un document ActiveX différent (si on considère qu'elles ne contiennent pas de code qui viole les exigences des documents ActiveX). Vous pourrez lier les documents avec des liens hypertexte en utilisant des instructions HTML, mais chaque feuille deviendra un document ActiveX indépendant. L'assistant de migration ne convertit pas vraiment la totalité de l'application en un document ActiveX unique si l'application contient plusieurs feuilles.

Q Quels types d'applications ne sont pas convertis par l'assistant Migration de document ActiveX ?

R L'assistant Migration de document ActiveX convertit la plupart des applications Visual Basic, à l'exception de celles qui contiennent des objets OLE incorporés, car OLE est une technologie ancienne non supportée par les navigateurs Web. En outre, certains contrôles et certaines commandes de communications avancés peuvent ne pas fonctionner comme attendu après migration. Mais la plupart des applications Visual Basic seront converties sans problèmes.

La limitation d'un document ActiveX par feuille, décrite à la question précédente, ne pose un problème que si une feuille de l'application utilise les méthodes `Hide`, `Show`, `Load` ou `Unload` pour masquer ou afficher une autre feuille. L'assistant met ces méthodes en commentaires, comme la commande `End`, car une application document ActiveX ne se termine pas comme une application habituelle, elle reste en fait active jusqu'à ce que l'utilisateur change de page ou ferme le navigateur Internet.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités, ainsi que des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se trouvent à l'Annexe A.

Quiz

1. Que fait de l'URL que vous fournissez l'application de navigation Web que vous générez dans l'assistant Création d'application ?
2. Les utilisateurs de votre application doivent utiliser Internet Explorer comme navigateur Web pour que le contrôle de navigation Web Visual Basic fonctionne. Vrai ou faux ?
3. Vous devez utiliser Internet Explorer comme navigateur Web pour que le contrôle de navigation Web Visual Basic fonctionne. Vrai ou faux ?
4. Qu'est-ce que l'encapsulation ?
5. Quel service en ligne est supporté par certains contrôles Visual Basic ?
6. Quelle est la différence entre un intranet et l'Internet ?

7. Quelle est la différence entre un document ActiveX et une application Visual Basic classique ?
8. A quoi sert Java ?
9. Quel langage de script fonctionne avec HTML pour charger et exécuter des documents ActiveX ?
10. Comment pouvez-vous convertir des applications existantes en documents ActiveX ?

Exercices

1. Si vous disposez de Microsoft Office Professionnel, vous pouvez utiliser le Classeur Office pour contenir des documents ActiveX ! Essayez en utilisant le document ActiveX calculette que vous avez créé dans la leçon d'aujourd'hui.
2. Sélectionnez une application qui contient plusieurs feuilles, comme le projet exemple Controls. Convertissez cette application en un document ActiveX.

Chapitre 20

Fournir de l'aide

Ce chapitre explique comment aider les utilisateurs. Lorsque vous en aurez terminé, vous saurez ajouter un système d'aide à une application afin que l'utilisateur puisse en lire la documentation en ligne. Le système d'aide est dit en ligne non pas parce que vous envoyez l'utilisateur sur l'Internet, mais parce que les informations sont immédiatement disponibles ; il n'a pas à se reporter à un manuel imprimé. Le système d'aide imite le système d'aide de presque toutes les applications Windows. L'utilisateur n'a pas à faire d'efforts particuliers d'apprentissage pour savoir comment utiliser l'aide en ligne de vos applications.

Cette leçon commence par vous apprendre comment ajouter des info-bulles aux contrôles. L'aide "Qu'est-ce que c'est ?" peut également être ajoutée une fois que vous maîtrisez la création des fichiers d'aide.

Vous apprendrez dans ce chapite :

- les info-bulles et l'aide "Qu'est-ce que c'est ?" ;
- la préparation des fichiers d'aide HTML, ;
- comment choisir entre l'aide HTML et l'aide Windows classique ;
- comment créer et formater le texte des fichiers d'aide, ;
- les exigences du format RTF ;
- comment lier les messages d'aide aux contrôles, ;
- comment utiliser le contrôle Boîte de dialogue commune pour commander les écrans d'aide ;
- comment les ID de contexte de l'aide pointent des sujets particuliers.

Il existe plusieurs méthodes pour ajouter de l'aide en ligne aux applications Visual Basic. La leçon d'aujourd'hui vous en enseigne deux principales. Tout d'abord, l'aide en ligne que vous trouvez encore dans la plupart des applications Windows, nommée WinHelp. Depuis Visual Basic 6, vous pouvez également ajouter de l'aide HTML à vos applications. Une partie de la leçon l'explique. Tous les utilisateurs ne disposant pas de navigateurs Web capables d'afficher l'aide HTML, vous ne pourrez garantir la compatibilité de vos applications avec leurs systèmes qu'en ajoutant l'aide en ligne standard.

Info-bulles et aide "Qu'est-ce que c'est ?"

Les *info-bulles* sont faciles à ajouter à divers objets, et l'aide particulière "Qu'est-ce que c'est ?" repose fortement sur le fichier d'aide. Vous pouvez ajouter des info-bulles à un contrôle quand vous insérez ce dernier dans une feuille, en renseignant la propriété `ToolTipText`. Quand l'utilisateur positionne ensuite le pointeur de la souris sur le contrôle, l'info-bulle s'affiche au bout d'un bref délai. Et, dès que vous disposez d'un fichier d'aide détaillé, l'ajout de l'aide "Qu'est-ce que c'est ?" est simple.

Les info-bulles sont les descriptions d'aide qui s'affichent quand vous laissez le pointeur de la souris sur un contrôle.

L'aide "Qu'est-ce que c'est ?" est une aide contextuelle (signifiant qu'elle dépend de l'endroit d'où l'utilisateur l'a demandée) qui décrit le contrôle. L'utilisateur clique sur le point d'interrogation "Qu'est-ce que c'est ?" dans le coin supérieur droit de la fenêtre ou sélectionne le menu Aide, Qu'est-ce que c'est ? ; le curseur se transforme en un point d'interrogation. Tout contrôle sur lequel l'utilisateur clique alors génère une aide.

Proposez une aide "Qu'est-ce que c'est ?" lorsque l'écran de votre application contient de nombreux éléments qui peuvent troubler les nouveaux utilisateurs. Ils peuvent cliquer sur "Qu'est-ce que c'est ?" dans la barre d'outil ou sélectionner l'option de menu correspondante, puis cliquer sur l'objet de l'écran pour lequel ils veulent en savoir plus.

Outre les info-bulles, vous pouvez proposer une aide "Qu'est-ce que c'est ?" pour donner aux utilisateurs des informations supplémentaires sur vos applications. La Figure 20.1 montre la fenêtre d'aide "Qu'est-ce que c'est ?" qui s'affiche après que l'utilisateur a choisi Aide, Qu'est-ce que c'est ? et qu'il a cliqué sur Parcourir. Le

pointeur en point d'interrogation reprend une forme normale dès que la boîte d'aide s'affiche.

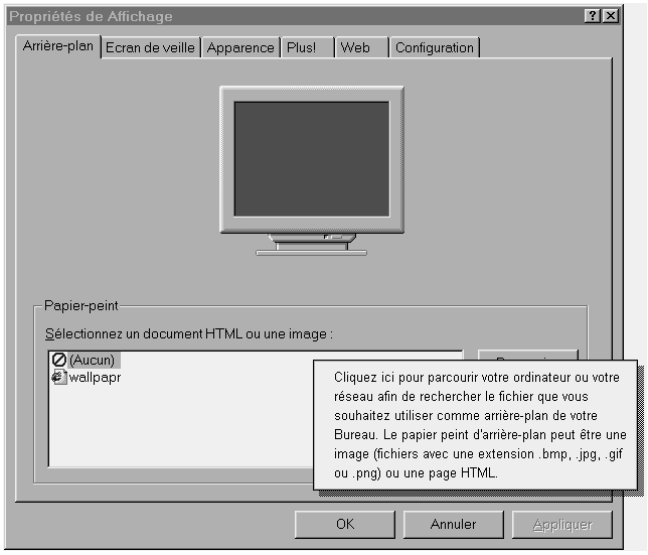


Figure 20.1
L'utilisateur peut demander "Qu'est-ce que c'est ?"

Faites correspondre l'aide "Qu'est-ce que c'est ?" aux objets. La Figure 20.1 utilise la page d'aide la plus proche disponible pour le bouton Parcourir dans la configuration des pages d'aide. Le processus est décrit dans la suite de cette leçon.

Cette étude simultanée des info-bulles et de l'aide "Qu'est-ce que c'est ?" s'explique par leur ressemblance d'un point de vue utilisateur. Ce dernier doit laisser le curseur sur un contrôle pour lire l'info-bulle, ou cliquer sur le déclencheur "Qu'est-ce que c'est ?" pour obtenir de l'aide sur ce contrôle.

Malgré ces ressemblances, vous pourrez être surpris de la complexité inhérente à l'ajout de l'aide "Qu'est-ce que c'est ?" par rapport aux info-bulles (qui sont très simples). Le reste de cette section décrit comment ajouter des info-bulles et la partie finale décrit comment ajouter l'aide contextuelle (une fois que vous aurez appris comment créer des fichiers d'aide à intégrer dans l'aide en ligne).

La simplicité de l'ajout des info-bulles à tout contrôle placé sur la feuille peut vous surprendre. Il suffit pour cela de saisir le texte dans la propriété `ToolTipText` de l'objet. Visual Basic s'occupe de tout le reste. Cherchez, dans les exemples, l'application de

bloc-notes MDI nommée Mdinote. Lorsque les programmeurs de Microsoft ont créé cette application, ils ont ajouté des info-bulles à presque tous les contrôles de chaque feuille. Vous pouvez sélectionner un contrôle au hasard et lire la propriété `ToolTipText` correspondante qui s'affiche quand le pointeur de la souris reste dessus.

Faire
Prenez l'habitude de renseigner le texte de l'info-bulle chaque fois que vous ajoutez un contrôle dans une feuille d'une application, vu la simplicité de la manipulation. Il y a moins de chances que vous ajoutiez les info-bulles plus tard, ajoutez-les donc immédiatement, quand le but de votre contrôle est encore frais dans votre esprit.

Adaptation de l'aide à une application

L'écriture d'un système d'aide en ligne peut être une tâche décourageante, mais vous pouvez emprunter celui de Windows dans vos applications. Tout ce que vous devez faire est d'écrire le texte de l'aide, puis d'utiliser le système d'aide de Windows pour afficher ce texte au bon moment et au bon endroit dans l'application. L'utilisateur n'a pas d'apprentissage à faire, car il sait déjà comment utiliser le système d'aide de Windows.

N'attendez pas d'avoir terminé votre application pour lancer la conception du système d'aide en ligne. Le meilleur moment pour rédiger le texte d'aide est le moment où vous concevez et créez l'application. C'est là que vous avez la meilleure connaissance du fonctionnement de l'application et que vous êtes le mieux armé pour écrire le type d'aide dont l'utilisateur a besoin.

Systèmes d'aide HTML

Si vous utilisez le système d'aide de Visual Basic, vous verrez l'Aide basée sur les fichiers d'aide *HTML*. En d'autres termes, l'aide s'affiche dans un format proche des navigateurs Web, avec des liens hypertexte. Vous pouvez cliquer sur Précédent pour afficher les écrans d'aides que vous avez déjà lus. Le panneau de gauche contient les thèmes du sommaire et lorsque vous double-cliquez sur une entrée, le panneau de droite affiche le contenu de la rubrique.

HTML signifie Hypertext Markup Language. C'est le langage de mise en forme des pages Web.

Vous pouvez ajouter ce type d'aide HTML à vos applications Visual Basic. C'est un nouveau type d'aide ; en effet, avant l'apparition des navigateurs Web, toutes les applications Windows utilisaient le même moteur, *WinHelp*. Ce dernier permet l'affichage de l'aide en ligne et utilise les liens hypertexte, mais dans un format différent des navigateurs Web (voir Figure 20.2). La version WinHelp du système d'aide des applications Windows est le même depuis plusieurs années. Les programmeurs prédisent que l'aide HTML sera le système standard de l'aide en ligne des applications Windows dans quelques années, mais WinHelp garde toujours une bonne avance.

WinHelp est le système d'aide en ligne utilisé dans la plupart des applications Windows, et qui sera peut-être remplacé un jour par le système d'aide HTML.

Figure 20.2
L'aide HTML utilise une fenêtre à deux volets ressemblant à l'Explorateur.

Malgré l'apparence moderne de l'aide fondée sur HTML, les utilisateurs de vos applications ne pourront pas obtenir d'aide en ligne s'ils ne disposent pas d'un navigateur Web installé sur leur système ou si leur navigateur est antérieur à Internet Explorer 4.0. Vous limiterez donc de façon considérable votre audience si vous exigez que tous les utilisateurs de vos applications utilisent Internet Explorer 4. Tant que la présence d'un navigateur Web n'est pas garantie sur la machine de l'utilisateur, il est préférable de s'en tenir au système d'aide WinHelp classique décrit dans la section suivante. En attendant, cette section explique quel sera votre futur si le monde informatique continue d'évoluer vers une interface basée sur les navigateurs.

La manière de connecter un fichier d'aide HTML à une application Visual Basic est la même que pour un fichier WinHelp : spécifiez le fichier d'aide HTML dans la propriété `HelpFile`. Vous pouvez en outre paramétrer la propriété `HelpContextID` pour tous les objets de la feuille pour qu'un sujet particulier d'aide contextuelle s'affiche s'il est sélectionné à l'écran quand l'utilisateur appelle l'aide.

La principale difficulté ne réside pas dans la connexion du fichier d'aide à votre application et à ses objets, mais dans la création de ce fichier. A partir de la section suivante, vous suivrez un exemple assez ennuyeux qui crée un fichier d'aide WinHelp qui, malgré sa longueur, est encore assez incomplet. Les écrans d'aide, même pour les applications simples, peuvent être nombreux et, plus vous offrez de liens hypertexte complets, plus votre travail de concepteur de fichiers d'aide devient pesant.

Les fichiers d'aide HTML ont une extension .CHM et les fichiers WinHelp, une extension .HLP.

La création d'un fichier d'aide, qu'il soit HTML ou WinHelp, nécessite souvent beaucoup plus d'efforts qu'il n'en mérite. Cependant, l'expérience que vous allez commencer à acquérir à partir de la section suivante est une bonne chose, car vous comprendrez mieux comment le texte d'un fichier d'aide interagit avec une application. Faites-vous plaisir dès que vous en aurez terminé avec cette leçon : courez chez votre marchand de logiciels et cherchez un outil de création d'aide plus automatisé. De nombreux logiciels utilitaires rendent la création beaucoup plus simple. De plus, les nouveaux outils de création d'aide génèrent l'aide HTML et WinHelp à partir des mêmes instructions.

Il existe d'autres outils — outre les outils de création d'aide disponibles actuellement — avec lesquels il est possible de créer des pages HTML utilisables comme fichiers d'aide. De nombreux traitements de texte actuels, comme Microsoft Word 97, ouvrent et enregistrent les fichiers au format HTML. Vous pouvez aussi, si vous n'avez pas accès à un programme de création d'aide au format HTML, utiliser un outil de conception de pages Web pour créer les fichiers HTML. Microsoft fournit une version de son programme de conception de pages Web, *FrontPage Express*, avec Internet Explorer. FrontPage Express peut aussi être téléchargé à partir du site Web de Microsoft.

Il existe des ouvrages complets qui expliquent comment créer des pages HTML, notamment les éléments de pages Web et d'aide, tels que les liens hypertexte et les graphiques incorporés.

L'aide RTF

Les sections suivantes expliquent comment créer et lier le fichier des sujets d'aide à votre application en utilisant la méthode traditionnelle WinHelp. Vous devez non seulement rédiger le fichier des sujets d'aide, mais aussi créer un projet d'aide que vous compilez ensuite pour donner un système d'aide lié à votre application. Le texte à insérer dans le fichier des sujets d'aide doit être au format RTF (*Rich Text Format*), que l'application peut utiliser pour incorporer des sauts hypertexte de sujet en sujet lorsque c'est nécessaire.

Un saut hypertexte est un lien vers un autre sujet dans le système d'aide.

Vous devez avoir accès à un traitement de texte capable de créer des fichiers RTF et qui supporte les notes de bas de page. Microsoft Word est certainement l'outil le plus répandu pour la création de fichiers d'aide.

Souvenez-vous qu'une bonne raison pour créer des fichiers d'aide WinHelp est qu'ils seront accessibles par tous les utilisateurs de votre application. Tous les exemplaires de Windows sont fournis avec un système d'aide qui s'appelle la *visionneuse de fichier d'aide (Windows Help Viewer)*, et qui est capable de gérer les fichiers source RTF WinHelp compilés comme c'est décrit ici. Cette visionneuse est, par contre, incapable de lire les fichiers d'aide HTML ; il faut pour cela un navigateur Web.

La visionneuse de fichier d'aide est un outil qui fait partie intégrante de Windows et permet d'afficher les fichiers WinHelp.

La compilation effectuée sur les fichiers d'aide n'a pas de rapport avec la compilation de l'application. Dans la leçon de demain, vous apprendrez comment compiler vos applications afin de les distribuer. Pour que votre application, compilée ou pas, accède à un fichier d'aide, il faut que ce dernier soit compilé en un fichier avec l'extension standard .HLP.

Préparer le fichier des sujets

Si vous utilisez Word pour créer le fichier d'aide, vous devez afficher les codes de formatage masqués. La plupart des gens préfèrent rédiger les documents sans afficher les codes, mais les sauts hypertexte exigent du texte masqué, que vous devez voir, même s'il est caché aux utilisateurs. Cliquez sur Afficher/Masquer (le symbole de paragraphe) sur la barre d'outils pour afficher les codes masqués.

Créer les sauts hypertexte

La majeure partie du fichier d'aide sera constituée de texte normal décrivant les sujets d'aide. Le texte d'aide n'impose pas de contraintes de formatage ; vous pouvez modifier la taille et le style des caractères comme vous le voulez. Cependant, les sauts hypertexte exigent un formatage spécifique pour permettre au système d'aide de reconnaître leurs mots clés et de savoir où se situe le sujet lié.

Plus vous ajoutez de références croisées entre les sujets d'aide à l'aide de sauts hypertexte, plus votre système d'aide sera utile à l'utilisateur. Quand vous ajoutez un saut hypertexte, les utilisateurs n'ont pas à utiliser un menu pour sélectionner les sujets qui peuvent les intéresser. Ils peuvent juste "sauter" directement au sujet qu'ils veulent lire.

Les exigences pour créer des sauts hypertextes sont les suivantes :

- Ajouter un soulignement double à toutes les phrases de saut hypertexte. Elles s'afficheront en vert dans la fenêtre d'aide de l'utilisateur. Avec Word, il faut sélectionner le mot ou la phrase, sélectionner Police dans le menu Format, puis choisir Double dans la liste déroulante Soulignement. Vous pouvez aussi utiliser la combinaison de touches Ctrl-Alt-U, ou personnaliser la barre d'outils et ajouter l'icône de double soulignement.
- Faire suivre le texte du saut hypertexte d'une balise nommée *chaîne de contexte*, qui contient le sujet de destination du saut et qui est formatée en texte masqué. N'insérez aucun espace entre le saut hypertexte et la chaîne de contexte. Assurez-vous que vous ne formatez que la chaîne de contexte en texte masqué (attention aux espaces, aux ponctuations ou aux marques de paragraphes). Pour masquer le texte, sélectionnez Format, Police, et cliquez sur la case Masqué. Vous pouvez aussi utiliser la combinaison de touches Ctrl-Maj-U ou personnaliser la barre d'outils.

Une chaîne de contexte est une chaîne de caractères qui suit un saut hypertexte affiché dans le texte quand l'utilisateur demande une aide contextuelle.

- Séparer la page du sujet qui contient le saut hypertexte de la page de destination par un saut de page. Vous pouvez ajouter un saut de page dans le menu Insertion, en sélectionnant Saut, puis Saut de page (ou appuyer sur Ctrl-Entrée).
- Connecter le texte du saut hypertexte à la page de destination par au moins un des trois symboles personnalisés de note de bas de page :
- Afficher des descriptions et des définitions d'aide surgissantes par un soulignement simple des sujets à définir. Vous pouvez souligner le texte dans le menu format, en sélectionnant Police, puis Simple dans la liste déroulante Soulignement. Vous

pouvez aussi appuyer sur Ctrl-U ou ajouter une icône en personnalisant la barre d'outils.

Symbole	*Description*
#	Utilisé pour se connecter à la page de destination par la chaîne de contexte.
$	Utilisé pour placer le titre de la page de destination dans la zone de texte Rechercher du système d'aide et pour connecter le saut hypertexte au titre de la page de destination.
K	Utilisé pour se connecter à un sujet lors d'une recherche par mots clés. De nombreux sujets d'aide sont liés à leurs pages de destination par les trois symboles de bas de page. En d'autres termes, l'utilisateur peut sauter de sujet en sujet, les titres des sujets apparaissent dans la zone de texte Rechercher du système d'aide ; il peut aussi rechercher les sujets à l'aide de mots clés.

Il est beaucoup plus simple de montrer un exemple que de décrire un fichier d'aide. La section suivante illustre donc les différentes manières de configurer les sauts hypertexte et les destinations.

Si vous utilisez le symbole de note de bas de page `K` *pour désigner un sujet de recherche, ajoutez autant de sujets que vous le pouvez. Comme vous le verrez dans l'exemple, les notes de bas de page* `K` *contiennent souvent plusieurs entrées séparées par des points virgules. La note de bas de page suivante indique au système d'aide que le sujet doit apparaître dans quatre rubriques de l'index de l'aide :*

```
KModifier les options ; Commandes du menu ; Modifications ;
➥Modifier le Âtexte
```

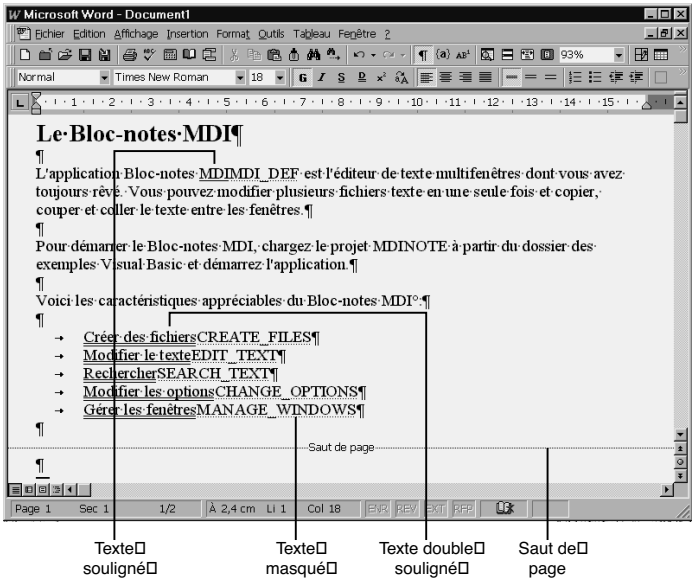
Créer un fichier d'aide

Vous avez, un peu plus tôt dans cette leçon, chargé le projet exemple de bloc-notes MDI (MdiNote.Vbp) pour voir comment fonctionnait l'aide par info-bulles. Vous allez créer un système d'aide auxiliaire pour cet exemple de programme MDI. L'application utilise les feuilles MDI pour gérer un petit éditeur de texte multifenêtres. Si l'application est relativement bien finie et va au-delà des capacités du Bloc-notes de Windows (car elle supporte plusieurs fenêtres, ce que ne fait pas le Bloc-notes de Windows), elle ne possède pas d'autre aide en ligne que les info-bulles.

Cet exemple utilise Microsoft Word pour la création des fichiers d'aide. Vous pourrez avoir à utiliser un autre traitement de texte, suivant le contenu de votre machine.

La Figure 20.3 montre un exemple d'écran d'ouverture de l'aide du Bloc-notes MDI dans Word. Les phrases en soulignement double sont les sauts hypertexte qui s'afficheront en vert dans la fenêtre d'aide de l'utilisateur. Le texte masqué étant affiché, il est souligné en pointillé.

Figure 20.3
L'écran d'ouverture de l'aide du Bloc-notes MDI contient des sauts hypertexte.

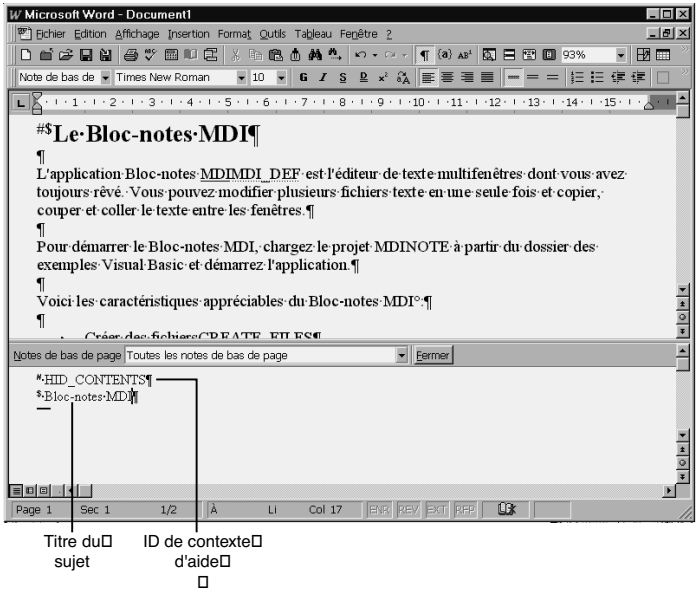


La Figure 20.3 montre six chaînes de contexte de phrases de saut : `MDI_DEF`, `CREATE_FILES`, `EDIT_TEXT`, `SEARCH_TEXT`, `CHANGE_OPTIONS`, et `MANAGE_WINDOWS`. Il devra donc y avoir au moins six autres pages après l'écran d'ouverture de l'aide. Ces pages sont connectées à leurs liens de saut hypertexte d'origine par un ou plusieurs des symboles de note de bas de page. Le premier saut, `MDI_DEF` sera une définition surgissante du terme *MDI*.

Le fichier d'aide doit avoir une valeur d'ID de contexte pour que l'application sous-jacente puisse faire référence à l'écran d'ouverture de l'aide si nécessaire. La Figure 20.4 montre deux notes de bas de page créées pour le texte d'accueil de l'aide. Pour ajouter une note de bas de page, vous devez déplacer le curseur devant le premier caractère du titre et, dans le menu Insertion, choisir Notes, Insérer notes de bas de page,

puis taper # comme symbole personnalisé pour indiquer l'emplacement de la page du saut hypertexte. Répétez ces étapes pour entrer la note $ pour le titre du lien au saut hypertexte. Les deux symboles des notes de bas de page s'affichent à gauche du texte et dans la fenêtre inférieure de note de bas de page. L'application peut utiliser l'ID de contexte d'aide pour référencer cet écran d'aide, et les outils de recherche du système d'aide afficheront le titre qui apparaît à gauche du symbole $.

Figure 20.4
La fenêtre d'aide complète s'affiche désormais lorsque l'ID de contexte ou le titre sont appelés.



N'utilisez pas le symbole de note de bas de page K dans la fenêtre d'ouverture de l'aide.

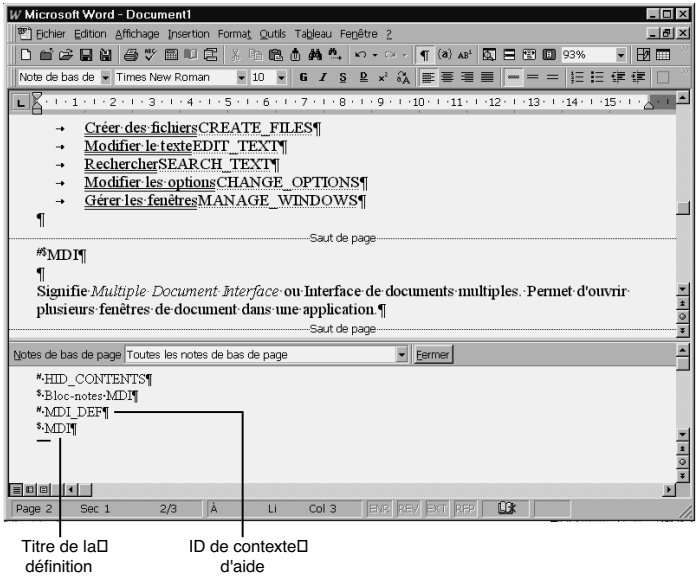
Les paragraphes suivants vous montreront que le document RTF contient un type d'ID de contexte différent de celui qu'attend Visual Basic. Vous devrez par conséquent faire correspondre les valeurs textuelles d'ID de contexte à des valeurs numériques pour qu'une application puisse utiliser l'aide contextuelle.

Les sauts hypertexte d'aide restants doivent maintenant avoir des pages d'aides correspondantes, avec leurs notes de bas de page pour pouvoir être connectées à l'écran d'ouverture. Le premier sujet d'aide à créer est la définition surgissante du terme MDI. La page qui suit l'écran d'ouverture doit contenir les informations suivantes :

- le titre MDI sur la première ligne ;
- une ligne de séparation ;
- la définition de MDI.

La note de bas de page de la Figure 20.5 complète la connexion entre cette page et le lien correspondant de la page d'ouverture en ajoutant un ID de contexte à la définition.

Figure 20.5
La définition surgira grâce à l'appel par soulignement simple.



Le fichier d'aide de la leçon d'aujourd'hui utilise, en général, le même ID de contexte d'aide (en majuscules) que le titre du sujet auquel il est lié, mais ce n'est pas une obligation.

La Figure 20.6, enfin, montre la première partie des sujets des sauts hypertexte restants de l'aide. La note de bas de page # connecte les sujets de saut hypertexte de la page d'ouverture aux pages suivantes.

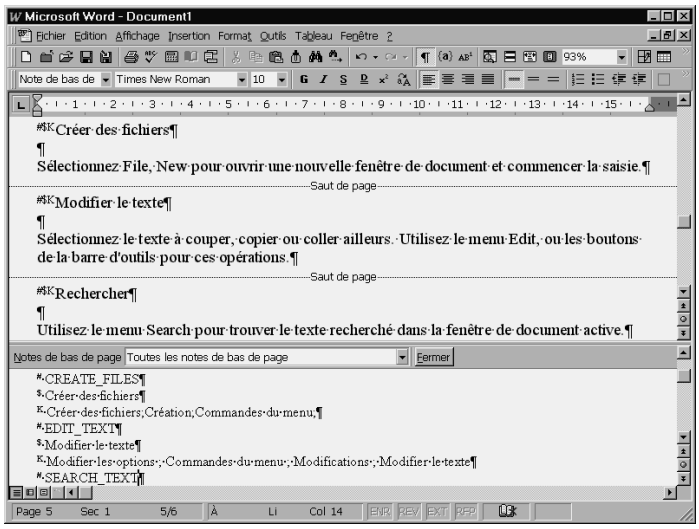


Figure 20.6
Les pages suivantes sont maintenant liées à la page d'ouverture de l'aide.

Toutes ces pages de sujets peuvent contenir des liens vers des pages supplémentaires (et entre elles) et aussi des définitions surgissantes.

Dès que vous avez achevé le fichier d'aide RTF, enregistrez-le. Sélectionnez bien le format Texte mis en forme (RTF) lors de l'enregistrement. Vous devez maintenant créer le fichier de projet d'aide en utilisant encore un autre type de fichier — Texte ASCII. Word peut enregistrer dans ce format ; vous pouvez même utiliser le Bloc-notes MDI pour créer ce fichier. Le fichier de projet suivant a été utilisé pour le fichier d'aide décrit :

```
[OPTIONS]
contents=HID_CONTENTS
title=Aide du bloc-notes MDI

[FILES]
MDINote.rtf
```

La section `[OPTIONS]` décrit l'ID de contexte de la page d'ouverture de l'aide et le texte de la barre de titre. La section `[FILES]` désigne le fichier d'aide à compiler (vous devrez indiquer un chemin si le fichier se situe dans un répertoire particulier). Entrez le nom du fichier d'aide RTF que vous venez de créer et d'enregistrer. Vous pouvez paramétrer d'autres options du projet d'aide à partir du compilateur d'aide.

Enregistrez le fichier de projet sous un nom de fichier quelconque (le nom du fichier de l'application est sans doute le meilleur choix). Utilisez l'extension .HPJ.

Vous devez exécuter le compilateur d'aide à partir du CD-ROM d'installation de Visual Basic, car il n'est pas installé avec Visual Basic. Pour exécuter le système d'aide, vous devez effectuer les opérations suivantes :

1. Insérez le CD-ROM d'installation de Visual Basic dans le lecteur.
2. Sélectionnez l'option Exécuter du menu Démarrage.
3. Exécutez le programme HCW.EXE situé dans le dossier \Common\Tools. La Figure 20.7 montre la fenêtre qui s'affiche, avec une astuce utile.

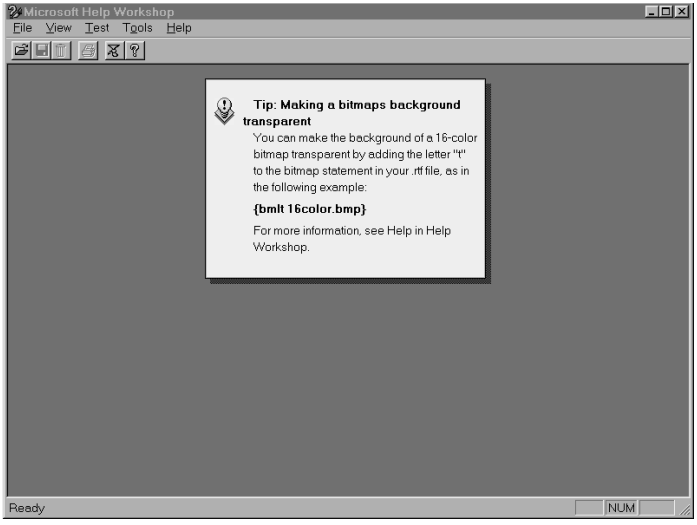


Figure 20.7
La fenêtre du compilateur d'aide propose une astuce pour démarrer.

Après avoir lancé le programme Microsoft Help Workshop, chargez le fichier de projet d'aide à compiler. Cliquez sur Compile dans la barre d'outils et acceptez toutes les valeurs par défaut, pour démarrer la compilation. Lisez les avertissements ou les erreurs qui peuvent être affichés à la fin de la compilation. Des avertissements qui n'ont pas d'incidence sur le fonctionnement du système d'aide se produisent souvent, mais il faut tenter de les éliminer complètement pour fiabiliser votre fichier d'aide dans toutes les situations. Si des erreurs sont présentes, le compilateur ne pourra pas compiler le fichier.

Une fois le système d'aide compilé, vous pouvez l'exécuter pour tester les entrées d'aide. Vous apprendrez comment le connecter à l'application dans la section suivante, mais vous pouvez déjà suivre ces étapes pour tester le fichier d'aide :

1. Démarrez l'explorateur Windows.

2. Recherchez le dossier contenant le fichier d'aide.
3. Cliquez du bouton droit sur le fichier d'aide et sélectionnez Ouvrir. Le système d'aide en ligne de Windows démarre et vous pouvez vérifier votre fichier.

La première fenêtre (à laquelle vous pourrez revenir à tout moment en cliquant sur Sommaire) affiche la page d'ouverture de l'aide. La page Index présente une liste complète de toutes les notes de bas de page K qui référencent les sujets d'aide.

Afficher le fichier d'aide

Une fois le fichier d'aide généré, vous devez y connecter l'application. Les ID de contexte d'aide associent les divers sujets aux contrôles et aux parties de l'application. N'oubliez pas non plus d'ajouter une option de menu Aide afin que l'utilisateur puisse appeler l'aide à tout moment.

Le nombre de connexions d'aide varie beaucoup d'une application à l'autre. Vous pouvez utiliser la profondeur du fichier d'aide et la complexité de l'application pour prévoir la quantité d'aide nécessaire aux utilisateurs. Le reste de cette section explique quelques méthodes de connexion du fichier d'aide aux applications. Visual Basic supporte de nombreuses connexions aux fichiers d'aide, mais cette section en décrit les plus courantes.

Boîte de dialogue Propriétés du projet

La boîte de dialogue Propriétés du projet, (voir la Figure 20.8) est le lien principal entre votre projet et le fichier d'aide. Elle garantit que le fichier d'aide est connecté lorsque l'utilisateur appuie sur la touche F1.

Vous n'avez pas à vous préoccuper de modifier la zone de texte ID de contexte d'aide au projet. Un ID de contexte d'aide saisi ici détermine la page d'aide qui s'affiche quand vous cliquez sur Aide dans la barre d'outils de cette application à partir de l'explorateur d'objets.

Contrôle Boîtes de dialogue communes

Vous avez pris soin de la partie la plus simple de la connexion au système d'aide en connectant la touche F1 à la fenêtre d'ouverture de l'aide. La fenêtre d'aide contiendra votre fichier d'aide, car les propriétés de votre projet pointent sur ce fichier. Pour ajouter une aide contextuelle pour que, lorsque l'utilisateur appuie sur F1, il obtienne de l'aide sur le contrôle ou la sélection de l'option de menu activé, vous devez effectuer quelques étapes de plus.

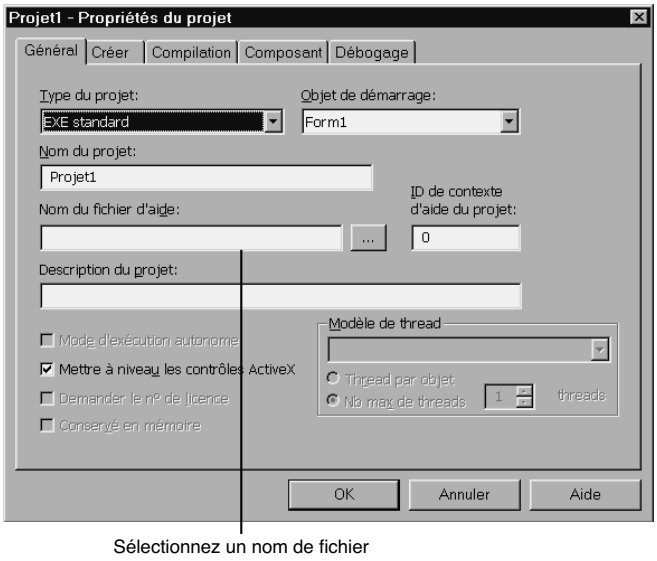


Figure 20.8
Connectez le fichier d'aide à votre projet.

Vous avez déjà vu, au Chapitre 9, comment utiliser le contrôle Boîtes de dialogue communes pour afficher différentes boîtes de dialogues comme Fichier Ouvrir ou Imprimer. Maintenant que vous avez créé un fichier d'aide, vous pouvez aussi utiliser ce contrôle pour afficher les écrans d'aide interactifs.

Lorsque vous placez le contrôle Boîtes de dialogue communes sur une feuille, paramétrez la propriété `HelpFile`, puis la méthode `ShowHelp`. Visual Basic lance le moteur d'aide de Windows qui interprète le fichier et propose les pages habituelles Sommaire, Index et Rechercher.

Vous pouvez limiter la capacité du moteur d'aide à ne montrer que l'onglet Sommaire, Index ou Rechercher en modifiant la propriété `HelpContext` *(voyez l'aide en ligne pour en connaître les valeurs). Mais l'usage veut qu'on propose, en général, les trois.*

Si vous proposez une aide contextuelle, vous devez indiquer au moteur d'aide quelle page doit s'afficher quand l'utilisateur sélectionne l'aide. Vous devez, pour cela, modifier le fichier de projet de l'aide et faire correspondre aux ID de contexte textuels des ID de contexte numériques.

Si l'utilisateur sélectionne l'aide contextuelle sur un contrôle où elle n'est pas correctement paramétrée, Visual Basic affiche la page Sommaire du fichier d'aide (la page d'ouverture).

Vous devez modifier à nouveau le fichier de projet et faire correspondre des nombres aux ID de contexte textuels. Pour cela, ajoutez une section `[MAP]` au fichier de projet. Si on considère le fichier d'aide MDINote.rtf et le fichier de projet associé décrit plus tôt, voici une modification possible :

```
[OPTIONS]
contents=HID_CONTENTS
title=Aide du bloc-notes MDI

[FILES]
MDINote.rtf

[MAP]
HID_CONTENTS 1
MDI_DEF 2
CREATE_FILES 3
EDIT_TEXT 4
SEARCH_TEXT 5
CHANGE_OPTIONS 6
MANAGE_WINDOWS 7
```

Assurez-vous qu'il n'y a pas deux numéros d'ID de contexte identiques. La correspondance peut partir de 1, mais de nombreux programmeurs Visual Basic réservent des séries de nombres pour représenter différents types de sujets d'aide. Par exemple, tous les boutons de commande pourront avoir des numéros compris entre 1 000 et 1 050. Les numéros n'ont pas à être en séquence. Recompilez le projet pour incorporer la nouvelle carte des informations dans le fichier d'aide.

Si vous manipulez Help Workshop, et que vous lisiez les écrans d'aide pour vous familiariser avec son fonctionnement, vous apprendrez comment la fonction Map rend la mise en correspondance des ID de contexte uniques plus rapide que leur modification par un éditeur de texte.

Vous devez localiser tous les contrôles et les feuilles auxquels vous voulez ajouter une aide contextuelle. Par exemple, dans le Bloc-notes MDI, vous pouvez afficher la feuille `frmFind` (qui supporte les options de recherche) et modifier la propriété `HelpContextID` du bouton de commande Find à `5`. Si vous vous limitez à cette manipulation, les utilisateurs verront l'écran d'ouverture de l'aide quand ils appuieront sur F1 dans l'application, mais si le bouton Find est activé, ils verront alors la page Rechercher. Bien sûr, vous devrez ajouter des affichages d'aide contextuelle à tous les autres contrôles et même à certaines feuilles, au bénéfice de l'utilisateur.

En ajoutant l'aide contextuelle aux contrôles, vous risquez de trouver d'autres zones de l'application qui demandent une aide. Vous aurez donc à faire plusieurs fois des ajouts au fichier RTF avant qu'il ne fournisse un support suffisant à l'application.

Lorsque vous ajoutez des pages d'aide, faites de votre mieux pour repérer tous les endroits de votre application où l'utilisateur risque d'en avoir besoin. Une aide contextuelle, chaque fois qu'elle est proposée, épargne à l'utilisateur une recherche dans l'index ou le sommaire.

Le Listing 20.1 montre le code d'affichage du sujet d'aide fourni par un ID de contexte particulier. Vous pouvez incorporer ce type de code dans un bouton de commande ou une option de menu pour proposer des sujets d'aide particuliers de votre fichier d'aide.

Listing 20.1 : Vous pouvez afficher une aide contextuelle

```
cdbHelp.HelpFile = "MDINote.hlp"  ' Pointe sur le fichier d'aide
'
' Vous pouvez proposer une aide spécifique sur un sujet
' particulier en pointant sur le numéro de la section
' [MAP] du fichier .HPJ (vos ID de contexte textuelles)
cdbHelp.HelpContext = 3  ' Pointe sur la section
cdbHelp.HelpCommand = cdlHelpContext  ' Demande contextuelle
cdbHelp.ShowHelp      ' Affiche l'aide contextuelle
```

Ajout d'aide "Qu'est-ce que c'est ?"

Maintenant que vous comprenez mieux comment créer un fichier d'aide, vous pouvez créer une aide "Qu'est-ce que c'est ?". Vous devez ajouter une page d'aide pour chaque fonction "Qu'est-ce que c'est ?" que vous voulez supporter. Une fois les pages ajoutées et connectées aux autres pages d'aide du système à l'aide des notes de bas de page personnalisées décrites plus avant dans cette leçon, vous devez faire correspondre des ID de contexte numériques aux pages. Le moteur de l'aide "Qu'est-ce que c'est ?" utilise les numéros d'ID de contexte pour déterminer la bonne fenêtre d'aide à afficher.

Le secret de l'aide "Qu'est-ce que c'est ?" tient en deux parties :

- Vérifier que la feuille supporte l'aide "Qu'est-ce que c'est ?" en paramétrant les propriétés `WhatsThisButton` et `WhatsThisHelp` à `True`.
- Entrer l'ID de contexte de la page d'aide dans la propriété `WhatsThisHelpID` de l'objet.

L'aide "Qu'est-ce que c'est ?" de l'application Bloc-notes MDI (si vous avez suivi les sections précédentes et créé un fichier d'aide avec des pages utilisables) ne demande que l'exécution de ces étapes :

1. Ouvrez la fenêtre de projet et double-cliquez sur la feuille `frmFind` pour afficher la boîte de dialogue Find.
2. Assignez la valeur `True` aux propriétés `WhatsThisButton` et `WhatsThisHelp`. Si vous affichez ensuite cette feuille en exécutant l'application, vous verrez le bouton "Qu'est-ce que c'est ?"en forme de point d'interrogation sur la feuille.
3. Répétez cette assignation pour les deux autres feuilles de l'application.
4. Recherchez dans la section `[MAP]` du fichier de projet d'aide les numéros d'ID de contexte et assignez-les aux diverses options du menu (utilisez le Créateur de menus) et aux objets feuilles qui peuvent nécessiter une description. Même si le fichier d'aide est loin d'être achevé, plusieurs des pages d'aide fonctionnent bien avec les objets, en particulier les éléments de la barre de menus.

En résumé

Ce chapitre vous a expliqué comment incorporer l'aide à vos applications. Plusieurs formes d'aides sont disponibles sous Visual Basic. En utilisant l'aide HTML ou le moteur d'aide WinHelp de Windows, vous pouvez créer un système complet de sauts hypertexte avec des pages interconnectées et des définitions surgissantes. Les utilisateurs peuvent obtenir des pages d'aide contextuelles particulières en appuyant sur F1. Le contrôle Boîtes de dialogues communes vous assiste dans la fourniture de l'aide, et permet aux utilisateurs de recevoir de l'aide en appuyant sur un bouton.

L'ajout d'aide contextuelle permet aux utilisateurs de trouver l'aide dont ils ont besoin. Vous pouvez assigner une aide contextuelle à différents objets pour qu'un texte particulier s'affiche quand l'objet est sélectionné et que l'utilisateur appuie sur la touche F1. La caractéristique d'aide contextuelle évite aux utilisateurs d'avoir à rechercher dans l'index chaque fois qu'ils ont besoin d'aide.

Deux fonctions d'aide simples qui peuvent être ajoutées rapidement sont les info-bulles et l'aide "Qu'est-ce que c'est ?". Les info-bulles sont très simples et ne demandent que l'ajout du texte à afficher dans la fenêtre Propriétés. Avant de pouvoir assigner une aide "Qu'est-ce que c'est ?", vous devez créer un fichier d'aide complet et assigner les divers ID de contexte numériques aux objets.

Le chapitre suivant conclut votre formation de 21 jours en vous montrant comment tester, déboguer et distribuer votre application.

Questions-réponses

Q Pourquoi ne puis-je pas créer les fichiers d'aide une fois que mon application est achevée ?

R Vous devez pouvoir le faire. C'est ce que vous avez fait aujourd'hui avec l'application Bloc-notes MDI. Cependant, vous saisissez mieux les tenants et les aboutissants de votre application au moment où vous la créez. Vous proposerez donc une meilleure aide si vous créez les info-bulles, les aides "Qu'est-ce que c'est ?" et les fichiers d'aide pendant la réalisation de votre projet. Vous pouvez laisser le traitement de texte ouvert dans une seconde fenêtre et y basculer (en utilisant la combinaison de touches Alt-Tab) lorsque vous voulez ajouter des éléments.

Q Pourquoi l'ajout d'aide dans les fichiers est-il si lourd ?

R La lourdeur vient du fait que vous devez garantir la connexion de tous les liens hypertexte au bon sujet, que tous les sujets qui doivent être traités le sont bien, et que vous utilisez un format correct dans le fichier d'aide RTF. Bien sûr, le compilateur d'aide repérera les fichiers d'aide mal formatés, et vous pourrez les déboguer. En fait, au cours de l'élaboration et du test de l'application, vous pouvez compiler aussi le fichier d'aide pour qu'il reste en phase. La création incrémentielle d'un fichier d'aide rend le processus général un peu moins lourd.

Il existe de nombreux outils de création d'aide, et vous pouvez les utiliser pour ajouter de l'aide aux programmes Visual Basic en évitant la lourdeur de la création des fichiers RTF ou HTML. N'utilisez l'aide HTML que si vous êtes certain que vos utilisateurs disposent d'un navigateur Internet compatible pour l'afficher.

L'utilisation de ces programmes visuels de conception d'aide rendra beaucoup plus souple la création de vos fichiers d'aide. En outre, vous éviterez nombre des erreurs de saisie commises naturellement avec l'approche RTF. Ces programmes vous permettent aussi de créer des diagrammes d'aide qui montrent comment se connectent les sauts hypertexte et réduisent donc la quantité de programmation RTF que vous devez effectuer pour créer le fichier d'aide.

Q Pourquoi ne pas utiliser ces outils et sauter ce chapitre ?

R Visual Basic supporte l'aide en ligne décrite dans cette leçon, mais aussi beaucoup d'autres — comme l'aide HTML. Vous pouvez utiliser des outils d'autres fournisseurs pour faciliter la création de l'aide et, si vous développez de nombreuses applications, vous les étudierez sûrement de plus près. Cependant, vous avez commencé avec Visual Basic et il contient tous les outils nécessaires, à part l'éditeur RTF, pour créer le support d'aide décrit aujourd'hui. De plus, tous les utilisateurs ne disposent pas de navigateur Web pour afficher l'aide HTML ; elle n'est donc pas destinée aux

applications universelles, sauf à exiger l'installation du navigateur pour que votre application fonctionne. En créant des fichiers d'aide par la méthode "à l'ancienne" étudiée aujourd'hui, vous en avez appris beaucoup sur le mode de fonctionnement du système d'aide et vous pouvez apprécier la complexité d'un système hypertexte. Une bonne partie de la lourdeur ne vient cependant pas de cette approche par fichier RTF, mais de la conception même du système d'aide. Plus il sera complet et complexe, mieux les utilisateurs apprécieront votre application. Mais des systèmes d'aide aussi complexes ne sont pas évidents à créer, c'est pourquoi il est utile de créer l'aide en même temps que l'application pour vous assurer que vous lui donnez l'aide qu'elle mérite.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités, ainsi que des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se situent à l'Annexe A.

Quiz

1. Quel format de fichier devez-vous utiliser pour créer le fichier d'aide en ligne ?
2. Comment les sauts hypertexte peuvent-ils améliorer un système d'aide ?
3. Quel est le symbole personnalisé de note de bas de page utilisé pour créer des définitions surgissantes liées au texte souligné de la page d'aide ?
4. Citez quelques caractéristiques du fichier de projet d'aide.
5. Une fois un fichier d'aide compilé, comment pouvez-vous l'attacher à une application pour qu'il s'affiche quand l'utilisateur appuie sur F1 ?
6. Comment connectez-vous l'aide contextuelle aux sujets du fichier d'aide ?
7. L'aide contextuelle utilise les ID de contexte textuelles du fichier d'aide. Vrai ou faux ?

8. Quelle est la différence entre l'aide "Qu'est-ce que c'est ?" et les info-bulles ?
9. Comment ajouter le bouton "Qu'est-ce que c'est ?" aux feuilles ?
10. Vous pouvez proposer l'aide "Qu'est-ce que c'est ?" sur les feuilles comme sur les objets de feuilles (*indice* : vérifiez les propriétés de la feuille). Vrai ou faux ?

Exercice

Ajoutez une aide "Qu'est-ce que c'est ?" à tous les objets du Bloc-notes MDI. Cette tâche peut sembler lourde (elle l'est un peu), mais vous arriverez vite à maîtriser le travail sur les aides "Qu'est-ce que c'est ?", les ID de contexte et les pages d'aide.

Chapitre 21

Distribution de vos applications

La leçon d'aujourd'hui montre comment tester, déboguer et distribuer vos applications Visual Basic. Une application n'est jamais achevée ; vous pouvez toujours y ajouter d'autres caractéristiques et, très souvent, des erreurs apparaissent bien après que vous pensez les avoir toutes extirpées. La maintenance sur le long terme fait donc partie intégrante du processus de programmation. Tout au long des leçons précédentes, ce didacticiel a proposé des astuces pour vous aider à mieux documenter votre code et à réduire les problèmes de maintenance.

Une des meilleures manières de le faire est de déboguer et de tester minutieusement vos applications avant de les distribuer. Cette leçon décrit quelques outils de débogage fournis par Visual Basic, comme des procédures de test qu'il peut être utile de faire subir à vos applications avant de les distribuer.

Nous voici à la fin de ce cycle de 21 jours. Une fois cette leçon terminée, considérez-vous comme lauréat de l'Université Visual Basic 6 avec un diplôme en Techniques de programmation et, plus important, prenez rang comme conseiller Visual Basic. Vous devrez alors pratiquer le développement autant que vous le pouvez pour affûter les compétences acquises au cours de ces leçons.

Vous apprendrez aujourd'hui :

- les types de bogues qu'un programme peut générer ;
- comment localiser les bogues en rédigeant le code, ;
- les nombreuses fenêtres de débogage, ;
- comment utiliser le mode pas à pas du débogueur pour localiser des parties spécifiques du programme durant l'exécution ;
- les divers points d'arrêt ;
- comment créer une routine d'installation de votre application ;
- comment utiliser l'assistant Empaquetage et déploiement, ;
- l'importance de l'installation de l'application.

Débogage et tests

Toutes les applications ont besoin d'être testées. Trop de bogues peuvent s'introduire durant les étapes de programmation. Lorsque vous testez une application, vous lui faites passer une batterie de cas. Pendant les tests, entrez des valeurs extrêmes et aléatoires dans tous les contrôles de saisie utilisateur pour vous assurer que l'application peut gérer des valeurs en dehors de la plage classique. Vous trouverez presque toujours des bogues durant la phase de test.

Le débogage se déroule en trois étapes :

1. Déterminer les bogues à problèmes et leur emplacement.
2. Corriger les bogues.
3. Tester à nouveau l'application pour s'assurer que les bogues sont éliminées.

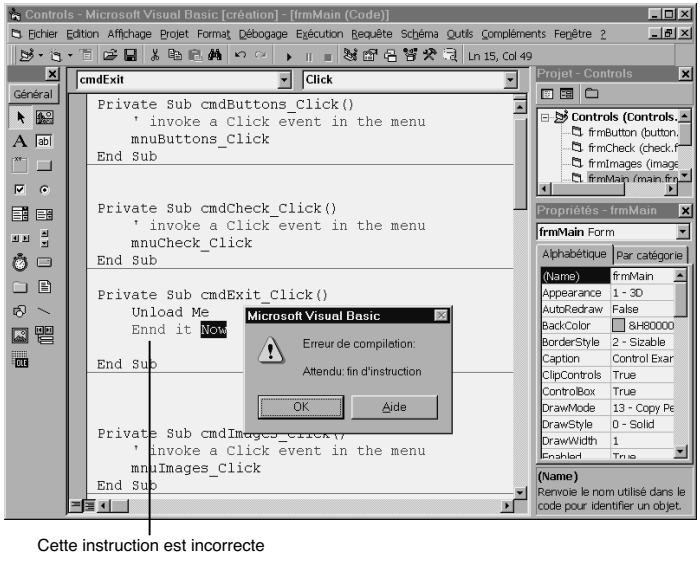
Les bogues vont du léger, comme une erreur d'orthographe ou d'alignement de texte, au grave, par exemple, une application qui termine la session Windows et cause une perte des données. Pour l'utilisateur, une bogue est tout ce qui ne correspond pas aux résultats attendus ou qui empêche l'application de s'exécuter.

Les programmeurs ont à faire face à de nombreux problèmes dans leur recherche des bogues. Il vous appartient de décider du moment où vous pensez avoir trouvé toutes les bogues qu'il vous était possible de détecter. Vous devez tester et tester encore pour garantir que les bogues ont été éliminées et ne se produisent plus. Une planification attentive avant, pendant et après le processus de codage aide à réduire le temps passé à déboguer les applications.

Il est préférable de développer et de tester vos applications dans l'environnement de développement de Visual Basic, qui contient des outils de débogage aidant à pister et à repérer les erreurs. Ce n'est qu'une fois satisfait du résultat des tests que vous pouvez compiler et distribuer vos applications aux utilisateurs.

Windows et le puissant environnement de développement de Visual Basic vous aident à localiser les erreurs. Quand vous exécutez une application, Visual Basic peut trouver une erreur durant la compilation ou la préparation à l'exécution d'un programme (comme un mot mal orthographié) et afficher un message d'erreur, comme celui que montre la Figure 21.1.

Figure 21.1
Visual Basic aide à repérer les bogues.



Si, lors de l'exécution d'une application, vous voyez un message de ce type avant que la première feuille s'affiche à l'écran, vous avez peut-être fait une erreur de syntaxe dans votre code. Le message d'erreur indique rarement *Erreur de syntaxe,* mais si elle s'est produite à la suite d'une erreur d'orthographe ou de grammaire, c'est bien le cas.

Une erreur de syntaxe est une erreur d'orthographe ou de grammaire dans un langage de programmation.

Remarquez que Visual Basic ne s'est pas contenté, à la Figure 21.1, de signaler l'erreur, mais qu'il en a aussi indiqué l'emplacement dans la fenêtre de code. Même si cette dernière est fermée lors de l'exécution du programme, Visual Basic met l'erreur en surbrillance. Le problème ici est une instruction `End` incorrecte. Après avoir corrigé l'erreur de syntaxe, vous pouvez cliquer sur Exécuter dans la barre d'outils pour reprendre l'exécution à partir de l'erreur corrigée.

Si la case Vérification automatique de la syntaxe est cochée dans l'onglet Editeur de la boîte de dialogue Options (sélectionnée par Outils, Options), Visual Basic vérifie les erreurs de syntaxe pendant que vous saisissez les instructions dans la fenêtre de code. Certains programmeurs préfèrent cependant avoir plus de liberté au moment de la conception pour pouvoir éparpiller ici et là des instructions partielles qu'ils complètent plus tard. Mais ce code incomplet peut conduire à des erreurs si vous n'êtes pas attentif : vous pouvez oublier de corriger une instruction. Il y a pourtant des moments où il est souhaitable de compléter plus tard les trous, par exemple s'il faut d'abord vérifier avec l'utilisateur une réponse à une question de conception.

Si vous désactivez la vérification automatique de la syntaxe, Visual Basic ne vérifie plus les erreurs de programmation, telle qu'une parenthèse manquante, tant que vous n'exécutez pas le programme. De toutes façons, Visual Basic localise ce type d'erreur par un message (voir Figure 21.1), mais l'option Vérification automatique de la syntaxe vous laisse le choix du moment où vous voulez être prévenu.

Des erreurs plus complexes apparaissent lors de l'exécution de l'application. Une erreur de syntaxe est facile à détecter, Visual Basic le faisant pour vous. Une erreur d'exécution est plus difficile à repérer et à corriger. Voyez l'erreur de la Figure 21.2, qui met en cause la logique du programme. Il n'y a pas de message d'erreur, mais dans le champ où le nom doit s'afficher, on trouve une adresse. A l'évidence, un champ d'adresse a été chargé à la place d'un champ nom. Visual Basic ne sait pas que c'est anormal, car il se contente de suivre les ordres du programmeur, même s'il en résulte des erreurs de logique.

Une erreur de logique repérée demande l'arrêt du programme. Visual Basic ne sait pas la reconnaître et interrompre l'exécution comme il le fait avec les erreurs de syntaxe) Vous devez ensuite localiser le problème.

Pour cela, vous devez rechercher dans le code de votre programme l'endroit où cette erreur de logique peut se situer, puis la corriger. Si le problème concerne l'apparence d'une feuille ou d'un contrôle à l'écran, vous devez rechercher toutes les références à cet objet. L'Explorateur d'objets peut souvent vous aider à trouver le code particulier attaché à un objet.

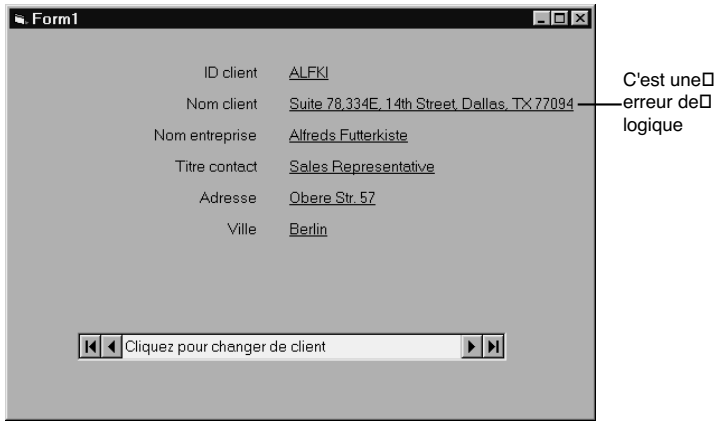


Figure 21.2
Visual Basic ne peut pas repérer des erreurs de logique.

Visual Basic peut détecter certaines erreurs de logique si elles résultent d'une impossibilité. Par exemple, la Figure 21.3 montre ce qui se passe si un programme demande à Visual Basic de diviser un nombre par zéro. Cette opération n'étant pas mathématiquement finie, Visual Basic est incapable de faire ce calcul, même s'il n'y a pas d'erreur de syntaxe. Il interrompt alors l'exécution et affiche une description de l'erreur dans une boîte de message.

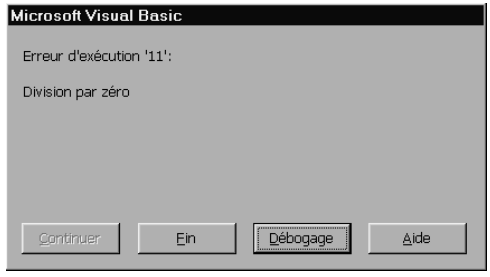


Figure 21.3
Certaines erreurs de logique demandent à Visual Basic d'effectuer quelque chose d'impossible.

Dès que Visual Basic réalise que le programme demande une tâche impossible, il affiche la fenêtre de code et signale l'endroit approximatif où la division par zéro s'est produite. Vous pouvez cliquer sur Aide dans la boîte de dialogue pour avoir une aide supplémentaire concernant le message. Cliquez sur Fin pour terminer l'exécution ou sur Débogage pour entrer dans le mode débogage de Visual Basic.

Remarquez que la division par zéro génère le code d'erreur 11 (voyez le message d'erreur de la Figure 21.3). Vous pouvez tester les erreurs dans l'objet système `Err.Number`*. Si vous soupçonnez qu'un calcul peut entraîner une division par zéro, du fait d'une donnée manquante, vous pouvez dérouter cette erreur par une instruction* `On Error Goto`*. Si* `Err.Number` *est à 11, vous pouvez informer l'utilisateur qu'il manque une donnée dans la feuille au lieu de le laisser dubitatif devant un message d'erreur.*

Le débogueur

L'environnement de développement de Visual Basic comprend un outil de débogage qui permet d'effectuer les tâches suivantes.

- Analyser le contenu des variables lors de l'exécution.
- Interrompre le programme sur n'importe quelle instruction et repartir quand vous êtes prêt.
- Positionner des *points d'arrêts* dans le code pour interrompre automatiquement l'exécution du programme.
- Modifier le contenu des variables en cours d'exécution pour tester l'application.
- Ajouter une variable *espion* qui interrompt l'exécution du programme quand elle reçoit une valeur ou une plage de valeurs particulière.
- Sauter des instructions que vous ne voulez pas exécuter durant un test.
- Utiliser la fenêtre des sorties de l'objet `Debug` pour imprimer des valeurs durant l'exécution d'un programme. La fenêtre de débogage permet de capturer des sorties, comme des valeurs de variables, sans déranger la fenêtre feuille normale.

  Vous pouvez entrer dans le mode débogage et avoir accès à toutes les fonctions du débogueur (situées surtout dans le menu Débogage) quand vous :

  – appuyez sur Ctrl-Break pour interrompre l'exécution du programme ;

  – recevez une boîte de message d'erreur à l'exécution ;

  – atteignez à l'exécution un point d'arrêt positionné ;

  – cliquez sur une instruction du programme, puis à partir du menu Débogage, choisissez Exécuter jusqu'au curseur pour exécuter normalement le programme. Visual Basic l'interrompt et entre en mode débogage dès que l'exécution atteint le curseur.

Positionner des points d'arrêt

Un des points d'arrêt les plus simples est l'exécution jusqu'au curseur. Pour le tester, chargez l'application Controls située dans le dossier des exemples. La feuille de test des boutons, `frmButton`, modifie un feu tricolore quand l'utilisateur clique sur le bouton de commande. Si vous suspectez le code de ne pas changer correctement le signal, vous pouvez cliquer sur la première instruction exécutable de la fonction `ChangeSignal()` dans le module standard et sélectionner Exécuter jusqu'au curseur dans le menu Débogage. (Ne sélectionnez pas un commentaire, car on ne peut pas y positionner de point d'arrêt.) Le programme démarre normalement, il s'arrête au point d'arrêt et met la ligne en surbrillance.

Ce n'est pas une interruption définitive. Jusqu'à ce point, toutes les variables du programme ont été initialisées, le code s'est exécuté et les résultats sont disponibles. Si une sortie se produit avant que la position du curseur soit atteinte (ce qui est le cas ici avec l'affichage de la feuille), vous en voyez normalement le résultat. Le programme, comme l'indique la barre de titre de Visual Basic, est en état d'*arrêt*. La ligne surlignée en jaune est la ligne où se situait le curseur quand vous avez choisi Exécuter jusqu'au curseur dans le menu Débogage.

Le Listing 21.1 montre la procédure où cet exemple particulier s'est arrêté.

Listing 21.1 : Vous pouvez analyser les procédures individuelles à un point d'arrêt

```
 1:  Private Sub ChangeSignal()
 2:      ' Check to see what color the light is, and then change
 3:      ' it to the next color. The order is green, yellow,
 4:      ' and then red.
 5:      If imgGreen.Visible = True Then
 6:          imgGreen.Visible = False
 7:          imgYellow.Visible = True
 8:      ElseIf imgYellow.Visible = True Then
 9:          imgYellow.Visible = False
10:         imgRed.Visible = True
11:    Else
12:         imgRed.Visible = False
13:         imgGreen.Visible = True
14:     End If
15: End Sub
```

Le feu ne fonctionne peut-être pas, car plus d'une couleur est affichée en même temps. Vous pouvez consulter les valeurs en cours des propriétés Visible des trois signaux possibles (`imgGreen`, `imgYellow` et `imgRed`) pour vous assurer qu'un seul est à `True` lorsque la procédure démarre.

Voir le contenu d'un contrôle (ou même d'une variable) n'a jamais été aussi simple. Comme le montre la Figure 21.4, tout ce que vous avez à faire est de positionner le pointeur de la souris dessus.

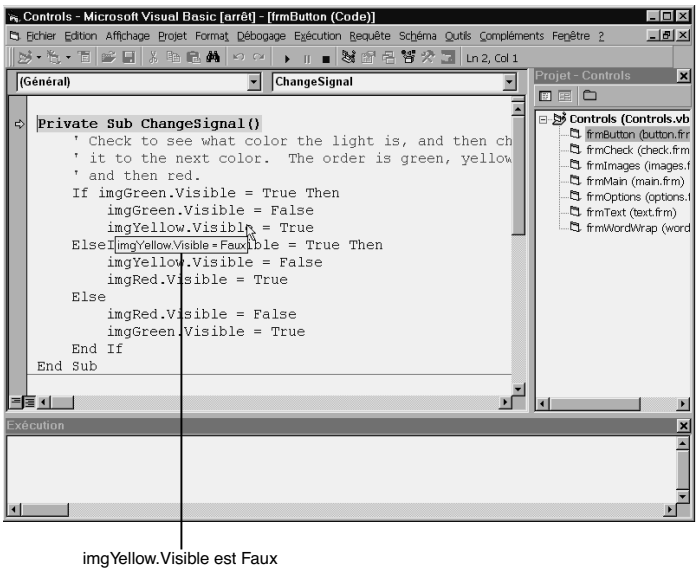


Figure 21.4
Visual Basic affiche toutes les valeurs de contrôles et de variables quand le programme s'arrête avec le débogueur.

En vérifiant les trois valeurs et ne trouvant qu'une seule `True`, vous déterminez que le problème ne réside pas dans la procédure `ChangeSignal()`. Vous devrez rechercher plus avant dans le code pour voir où les signaux ont été mélangés (c'est une supposition pour notre exemple). En arrêtant l'exécution à différents endroits et en analysant les variables et les contrôles, vous pouvez déterminer si les valeurs sont ce qu'elles devraient être.

Retracer vos pas

Avant d'aller plus loin, affichez la barre d'outils Débogage : dans le menu Affichage, choisissez Barres d'outils, puis Débogage. (Vous pouvez la laisser flotter ou la placer dans la bande des barres d'outils.) Lorsque vous avez besoin de savoir comment le programme est parvenu au point d'arrêt, vous pouvez utiliser une des fonctions de débogage les plus utiles : la *Pile des appels*. Cliquez sur le bouton correspondant de la barre d'outils. La boîte de dialogue Pile des appels s'affiche et montre le cheminement de votre programme, procédure par procédure, jusqu'à sa position actuelle (voir Figure 21.5).

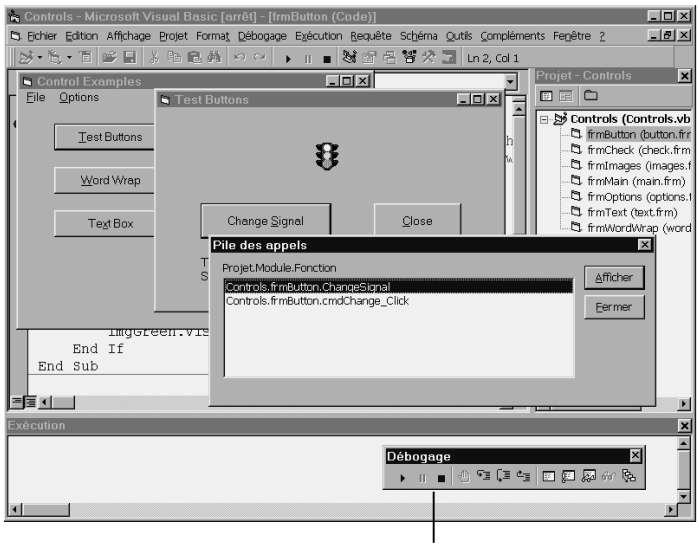


Figure 21.5
Utilisez la boîte de dialogue Pile des appels pour suivre les procédures du programme qui sont exécutées.

Si vous voyez une entrée de pile libellée `[<Code Non-Basic>]`, l'exécution s'est produite à partir d'une autre source, comme cela se passe quand le code fait des appels à l'API Windows.

L'API Windows (Application Programming Interface) est un ensemble de procédures internes à Windows que vous pouvez appeler depuis des langages comme Visual Basic ou Visual C++ quand vous avez besoin d'emprunter une routine ou de déclencher une fonction à partir de Windows. Le débogueur de Visual Basic n'a pas la capacité ni l'autorisation (de par les protections système de Windows) de suivre la trace des procédures du système d'exploitation.

Pour afficher une des procédures de la boîte de dialogue Pile des appels, double-cliquez sur l'entrée correspondante. Vous ne voyez pas alors uniquement des instructions, mais aussi des valeurs actives, car l'application est toujours en état d'arrêt. Vous pouvez donc lire les valeurs de tous les contrôles, les variables, et les constantes nommées dans les diverses procédures.

Avancer pas à pas dans le code

A tout point d'arrêt, vous pouvez cliquer sur Pas à pas détaillé pour exécuter l'instruction suivante du programme (même si c'est un appel à une autre procédure).

La barre d'outils Débogage contient trois boutons de pas à pas. Le Tableau 21.1 décrit comment les utiliser. Vous pouvez ne pas souhaiter avancer pas à pas sur toutes les instructions d'une application : ces trois boutons vous donnent une certaine liberté pour définir comment vous voulez que votre programme continue.

Tableau 21.1 : Les modes pas à pas déterminent comment votre application doit continuer

Mode pas à pas	*Description*
Pas à pas détaillé	Exécute l'instruction suivante. Même si elle se situe dans une autre procédure (ou qu'elle renvoie vers une procédure précédente), elle est exécutée et le curseur s'y place. Vous pouvez donc parcourir toute l'application instruction par instruction en appuyant continuellement sur F8.
Pas à pas principal	Exécute l'instruction suivante à moins que ce ne soit un appel à une autre procédure. Dans ce cas, la nouvelle procédure s'exécute entièrement, et l'exécution s'arrête à l'instruction qui suit l'appel de procédure.
Pas à pas sortant	Termine l'exécution de la procédure en cours, puis l'exécution s'arrête à la première instruction en dehors de la procédure.

Vous pouvez naturellement, à tout point d'arrêt, cliquer sur Continuer pour que l'exécution se poursuive de manière normale. Si des points d'arrêts sont positionnés plus loin, l'exécution s'y arrête. Sinon, le programme se comporte normalement, comme s'il n'avait jamais été interrompu.

Vous pouvez arrêter le mode Débogage à tout moment en cliquant sur Fin dans la barre d'outils de Visual Basic ou en sélectionnant Fin dans le menu Exécution.

Points d'arrêt multiples

Lors de l'exécution de votre application, vous pouvez souhaiter positionner en chemin des points d'arrêt pour vous permettre d'étudier les variables et les contrôles en cours d'exécution. Par exemple, si vous observez des problèmes que vous voulez pouvoir

analyser à la prochaine exécution, vous pouvez ajouter un point d'arrêt en cliquant sur Basculer le point d'arrêt dans la barre d'outils Débogage sur l'instruction surlignée. Vous pouvez positionner plusieurs points d'arrêt sur d'autres lignes partout dans le code en cliquant sur ce bouton. Quand vous atteignez un point d'arrêt (signalé par un surlignement rouge) positionné dans une session précédente, mais dont vous n'avez plus besoin, cliquez à nouveau sur Basculer le point d'arrêt sur cette ligne pour le supprimer. Vous pouvez aussi cliquer sur la gauche d'une instruction pour ajouter ou supprimer un point d'arrêt.

Vous ne pouvez positionner les points d'arrêts que sur les lignes exécutables. Il est impossible de positionner des points d'arrêts sur les instructions de déclaration de types de variables utilisateur ou de commentaires.

Fenêtre de débogage

A tout point d'arrêt, vous pouvez afficher la fenêtre de débogage pour travailler en dehors de l'environnement du programme. La fenêtre de débogage est souvent nommée *fenêtre Exécution*. Quand vous cliquez sur le bouton correspondant dans la barre d'outils Débogage, elle s'ouvre en bas de votre fenêtre de code (dans sa position rangée), ou elle peut flotter sur l'écran.

La fenêtre Exécution (nom de la fenêtre de débogage) est une fenêtre de l'environnement Visual Basic dans laquelle vous pouvez afficher des valeurs du programme et des messages durant l'exécution du programme. En envoyant des messages à la fenêtre Exécution, vous pouvez lire des messages d'avancement de l'exécution de l'application en utilisant la méthode `Debug.Print`*. Ces messages n'interféreront pas avec les sorties normales du programme.*

Vous pouvez taper toute instruction Visual Basic dans la fenêtre Exécution et voir le résultat immédiatement. Une des méthodes de débogage les plus courantes est `Print`, qui imprime les valeurs des variables et les propriétés de contrôles. `Print` envoie les sorties vers différents objets (pas uniquement vers une imprimante ou une feuille), dont la fenêtre Exécution. La Figure 21.6 montre la valeur d'un objet et illustre le fait que les valeurs avec lesquelles vous travaillez sont des valeurs actives, configurées par la partie de l'application exécutée jusqu'au point d'arrêt. Vous pouvez aussi imprimer le résultat d'une expression ou même modifier la valeur d'une variable en cours d'exécution. Dans ce cas, le reste du programme utilise cette nouvelle valeur à la place de la valeur assignée à l'origine par le code. Vous pouvez ainsi observer à chaud le résultat.

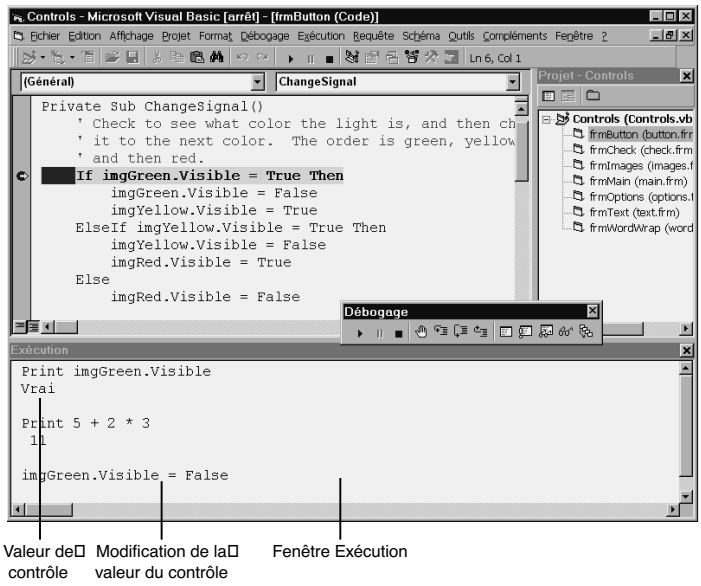


Figure 21.6
Utilisation de la fenêtre Exécution pour imprimer des valeurs et modifier les résultats.

Si la nature interactive de l'environnement de développement de Visual Basic rend ce type de sortie moins importante qu'elle ne l'était dans les environnements en mode caractères, vos programmes peuvent écrire directement dans la fenêtre Exécution en utilisant la méthode `Print` *de l'objet* `Debug`*. Si votre programme comprend une instruction comme* `Debug.Print cmdN6xt.Caption`*, la sortie est affichée dans la fenêtre Exécution, où vous pouvez la lire sans avoir à interférer avec les sorties normales de l'application dans la fenêtre feuille.*

Fenêtre Variables locales

Si vous cliquez sur Fenêtre Variables locales dans la barre d'outils Débogage, Visual Basic l'affiche (voir Figure 21.7). Elle montre les valeurs en cours de toutes les variables locales de la procédure en exécution (celle où se situe le point d'arrêt actif), les variables et les constantes globales. L'aspect le plus utile de cette fenêtre est sans doute son affichage de toutes les valeurs des contrôles de feuille. Vous pouvez agrandir ou rétrécir l'affichage pour voir tous les détails qui vous intéressent.

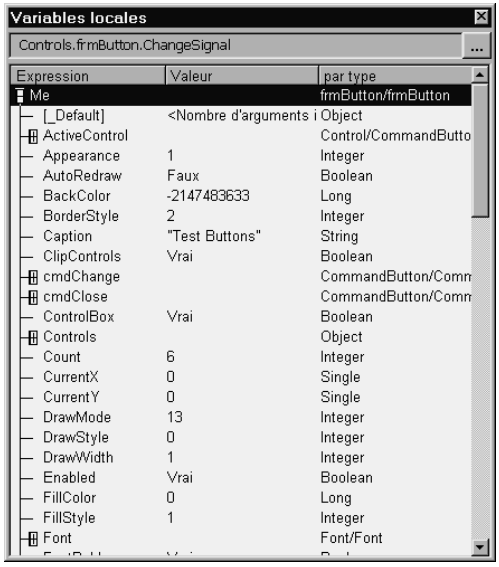


Figure 21.7
La fenêtre Variables locales montre toutes les variables, locales et globales, de la procédure en cours.

Si vous modifiez une variable locale dans la fenêtre Exécution, la valeur change aussi dans la fenêtre Variables locales.

Outre le nom et la valeur, la fenêtre Variables locales affiche le type de données de la variable ou du contrôle. Cliquez sur les points de suspension à droite du nom de la procédure en cours dans la fenêtre Variables locales pour afficher la fenêtre Pile des appels. Si vous cliquez sur une des procédures de la liste, la fenêtre affiche alors les variables locales de cette procédure.

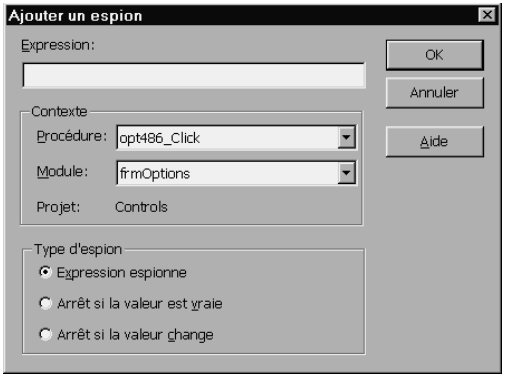
Fenêtre Espions

Au cours du processus de débogage, vous pouvez constater qu'une bogue ne se produit que si une variable a une certaine valeur. Le problème peut parfois ne pas pouvoir être pisté jusqu'à une instruction unique, vous devrez donc espionner une variable ou une expression sur toute la procédure. C'est la raison d'être de la fenêtre Espions, dans laquelle vous pouvez entrer des valeurs de variables ou d'expressions. Vous pouvez configurer dans la fenêtre des valeurs à espionner au moment de la conception ou de

l'exécution en cliquant sur Fenêtre Espions dans la barre d'outils Débogage. Utilisez l'une de ces deux méthodes pour ajouter des valeurs à la fenêtre Espions :

- A partir du menu Débogage, choisissez Ajouter un espion pour afficher la boîte de dialogue correspondante (voir la Figure 21.8).
- Cliquez avec le bouton droit sur la fenêtre Espions (qui doit être affichée en cliquant sur son bouton dans la barre d'outils Débogage), puis choisissez Ajouter un espion pour afficher la boîte de dialogue.

Figure 21.8
Ajouter des valeurs à espionner dans la fenêtre Ajouter un espion.



Quand vous ajoutez une expression à espionner, le contexte indique à Visual Basic la portée de l'espion (où il doit le surveiller). Visual Basic peut espionner une procédure, une feuille, un module ou tout le projet.

La zone Type d'espion vous permet de préciser la manière dont vous voulez que Visual Basic réagisse à la valeur espionnée. Il peut afficher les données de l'expression et arrêter l'exécution si une valeur est atteinte ou à chaque changement. Lors d'exécutions ultérieures, Visual Basic actualise la fenêtre Espions en fonction des valeurs espionnées.

Visual Basic comprend une fenêtre Espion express, qui permet d'ajouter des valeurs à espionner à la volée sur un point d'arrêt. Sélectionnez la variable, l'expression ou la propriété de contrôle, puis cliquez sur Espion express dans la barre d'outils Débogage. Vous pouvez aussi ajouter l'expression à la fenêtre Espion classique en cliquant sur Ajouter.

De nombreux programmeurs trouvent plus facile d'utiliser l'Espion express sur un point d'arrêt que de tenter de collecter toutes les valeurs à espionner au moment de la conception avec la boîte de dialogue Ajouter un espion.

Distribution de votre application

Votre application est créée, testée et déboguée ; reste à l'empaqueter pour la distribuer. Si elle est destinée à un usage personnel, vous n'aurez sans doute qu'à la compiler, puis à en copier les fichiers dans le dossier où l'exploiter. Vous pouvez utiliser la personnalisation du menu Démarrage dans les Propriétés de la Barre des tâches de Windows pour connecter l'application dans votre structure de menu Démarrage.

Pour que votre application soit utilisée par d'autres, vous devez automatiser l'installation afin que tous les fichiers du projet se situent à la bonne place et que le programme soit installé dans le menu Démarrage. Cette section explique comment distribuer une application en utilisant l'une des applications exemple fournies avec Visual Basic comme guide.

Compiler une application

Visual Basic simplifie la compilation de votre application. Le fichier compilé est un exécutable avec une extension .EXE. Tous les modules et feuilles apparentés sont groupés pour former le fichier exécutable. Même si des fichiers auxiliaires peuvent toujours être nécessaires — par exemple, un fichier de base de données Microsoft Access utilisé pour les données initiales — la plupart des fichiers de votre projet se combinent dans l'exécutable pour que la distribution en soit plus simple.

La distribution d'une application compilée est bien plus sûre que celle des sources. Si vous distribuez le code source (le projet et les fichiers apparentés), toute personne disposant de Visual Basic peut modifier votre travail. De plus, la plupart des utilisateurs ne pourront même pas l'utiliser, car ils ne disposeront pas de Visual Basic pour charger et exécuter le programme. Un fichier compilé est donc nécessaire pour permettre à tous d'utiliser l'application.

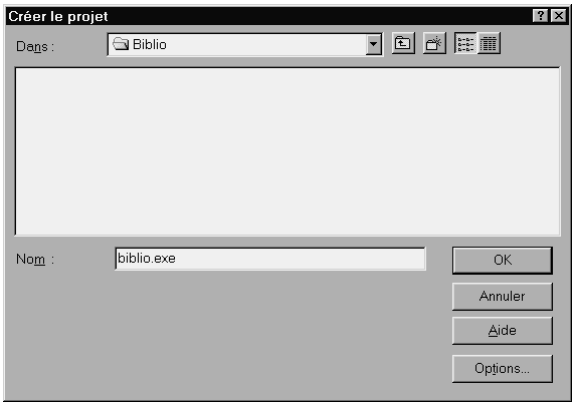
Une application compilée s'exécute beaucoup plus vite que dans l'environnement de développement de Visual Basic. Vous voulez que votre application s'exécute avec rapidité et souplesse sans que l'utilisateur ait à faire plus que le nécessaire ; le fichier compilé simplifie l'exécution de l'application.

Avant de compiler l'application, assurez-vous de l'avoir déboguée autant qu'il est possible. Vous ne pouvez pas déboguer une application compilée avec le débogueur de Visual Basic, car elle s'exécute hors de l'environnement de développement.

Une fois que vous pensez que votre programme s'exécute aussi bien que possible, choisissez Fichier, Créer. Visual Basic affiche la boîte de dialogue Créer le projet (voir

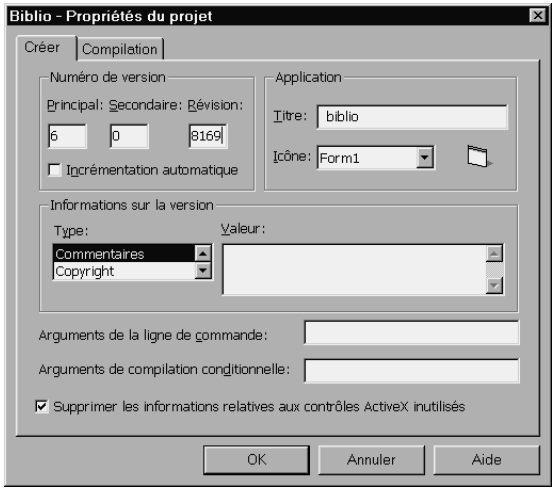
Figure 21.9). Sélectionnez le dossier de destination de l'application compilée. Visual Basic utilise par défaut le nom du projet comme nom d'exécutable.

Figure 21.9
Compilation de l'application à partir de la boîte de dialogue Créer le projet.



Avant de cliquer sur OK pour lancer la compilation, cliquez sur Options pour afficher la boîte de dialogue Propriétés du projet (voir Figure 21.10). Vous pouvez aussi y accéder en choisissant Propriétés dans le menu Projet. Elle vous permet de spécifier des informations de version dans l'application compilée, toujours utiles, surtout si vous prévoyez de diffuser plusieurs versions de votre application. Les numéros de versions et les informations de descriptions restent enregistrés dans le code source du projet, ce qui vous permet de gérer les différentes évolutions.

Figure 21.10
Configuration des options du projet compilé dans la boîte de dialogue Propriétés du projet.



La rubrique Icône désigne l'icône de l'application qui s'affiche dans le menu Démarrage de Windows et dans le bouton de la Barre des tâches. En général, vous laisserez le nom de la feuille principale dans ce champ, car la fenêtre Propriétés de la feuille contient une rubrique `Icon` dans laquelle vous pouvez choisir une icône pour la feuille et donc, pour l'application compilée.

Cliquez sur l'onglet Compilation pour afficher la page des options de compilation (voir Figure 21.11). Pour optimiser le projet compilé et qu'il s'exécute le plus vite possible, sélectionnez l'option Compiler en code natif. (La compilation en *P-Code* — ou *pseudo-code* — exige que l'utilisateur dispose d'une DLL Runtime Visual Basic — un fichier dans le dossier Système.) Le code natif est bien plus rapide et demande moins de fichiers, même s'il demande toujours la présence d'un fichier DLL.

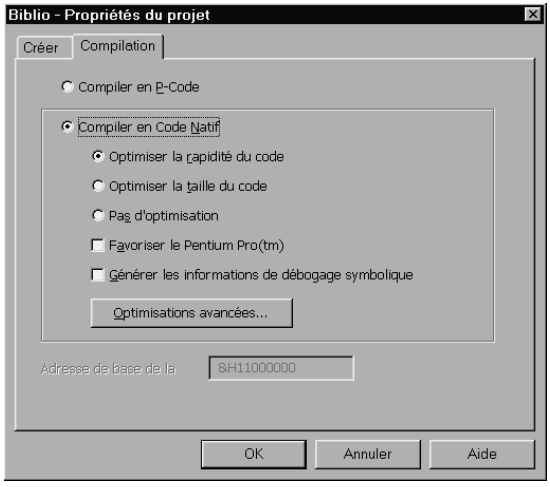


Figure 21.11
La page Compilation contient les options de compilation du projet.

Si vous sélectionnez des options dans la fenêtre qui s'affiche quand vous cliquez sur Optimisations avancées, vous perdrez l'avantage de certains contrôles à l'exécution, mais vous gagnerez en rapidité.

Quand vous cliquez sur OK, Visual Basic ferme la fenêtre des propriétés et compile le programme. Si aucune erreur de compilation ne se produit, Visual Basic crée le fichier .EXE (vous verrez l'état de la compilation dans le coin supérieur droit). Vous pouvez quitter Visual Basic et exécuter l'application en sélectionnant l'option Exécuter du menu Démarrage et en localisant le fichier .EXE. L'icône de feuille que vous avez sélectionnée s'affiche dans la Barre des tâches quand vous exécutez le programme.

L'assistant Empaquetage et déploiement

L'assistant Empaquetage et déploiement exécute de nombreuses tâches à votre place, dont :

- La compilation de l'application et la compression du fichier.
- La création d'un programme d'installation pour installer l'application.
- La détermination de la meilleure organisation des disquettes d'installation, la création des divers disques d'installation et le découpage des fichiers importants sur plusieurs disquettes. L'assistant indique à l'avance le nombre de disquettes nécessaires à l'installation.
- La copie de l'application compilée sur un disque dur pour que vous puissiez l'installer sur un réseau ou sur un graveur de CD-ROM.
- La configuration de l'application pour sa distribution sur l'Internet vers les utilisateurs d'Internet Explorer.

L'assistant génère une liste des différents fichiers nécessaires à l'installation. Un fichier Setup.exe unique n'est pas la seule chose à fournir dans la routine d'installation. Une application Visual Basic exige souvent des fichiers DLL et OCX, qui doivent résider sur le support d'installation (disquette ou disque) choisi avec le fichier de l'application compilée et le programme Setup.exe.

Avant de pouvoir exécuter l'assistant Empaquetage et déploiement, vous devez charger votre projet, débogué et compilé. L'assistant le compilera une dernière fois au cas où vous auriez fait des modifications de dernière minute depuis la dernière compilation.

L'assistant ne fait pas partie de l'environnement de développement Visual Basic. Vous devez le sélectionner dans le menu Compléments, Gestionnaire de compléments. Une fois chargé, vous pouvez le lancer. La Figure 21.12 montre la fenêtre d'ouverture.

La première option de l'assistant (certainement celle que vous choisirez la plupart du temps) crée une routine Setup.exe standard que l'utilisateur peut installer. L'assistant peut préparer l'installation sur un disque dur, des disquettes, un graveur de CD-ROM ou dans des fichiers CAB spéciaux que vous pouvez transmettre sur l'Internet pour une distribution en ligne. La seconde option envoie la routine d'installation vers un serveur Internet pour une installation à distance de l'application. L'assistant crée un fichier script qui décrit la routine d'installation. Lors de sessions ultérieures, vous pouvez modifier un script déjà créé ou le recréer à partir du projet d'origine. La troisième option vous permet de gérer les scripts d'installation.

La première option génère la forme la plus courante de routine d'installation, valable pour la plupart des applications. Après avoir cliqué sur le bouton, vous verrez la fenêtre de la Figure 21.13. A moins que votre application n'exige des fichiers de contrôles

ActiveX externes ou de bases de données, vous pouvez laisser la première option sélectionnée.

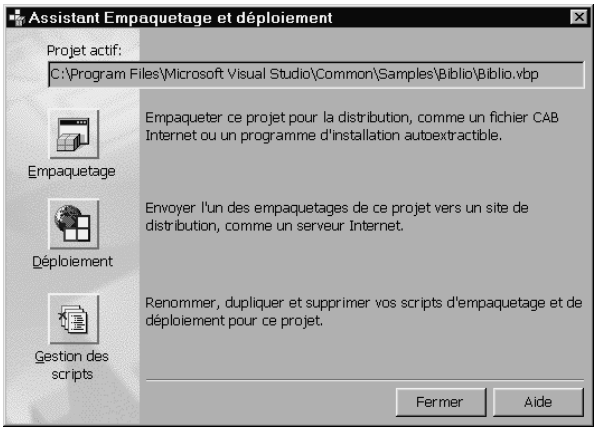


Figure 21.12
L'assistant Empaquetage et déploiement prépare votre application à la distribution.

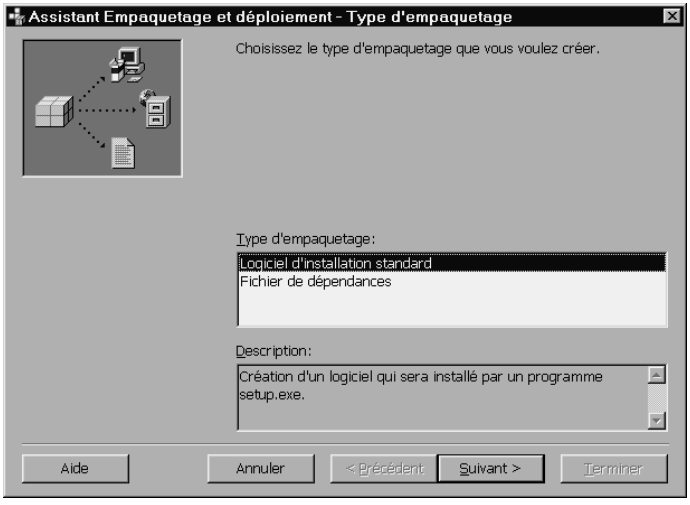


Figure 21.13
Détermination du type de produit à créer.

Pour installer un contrôle ActiveX, sélectionnez l'option Fichier de dépendances afin que l'assistant puisse collecter les bons fichiers dans l'ordre où l'application en a besoin.

Cliquez sur Suivant pour afficher la boîte de dialogue Dossier d'empaquetage, qui demande des informations de distribution. L'assistant doit savoir où assembler les fichiers d'installation. Choisissez un répertoire vide. Ainsi, vous saurez, quand l'assistant en a fini, que tous les fichiers de ce dossier sont ceux qu'il a générés.

Lorsque vous cliquez sur Suivant, l'assistant scrute votre projet pour déterminer les fichiers de programmes nécessaires à votre application. Si votre application contient des contrôles de données, il ne peut pas savoir quels pilotes de base de données sont requis. Vous verrez dans ce cas la boîte de dialogue de la Figure 21.14. Copiez le pilote requis dans la fenêtre de gauche.

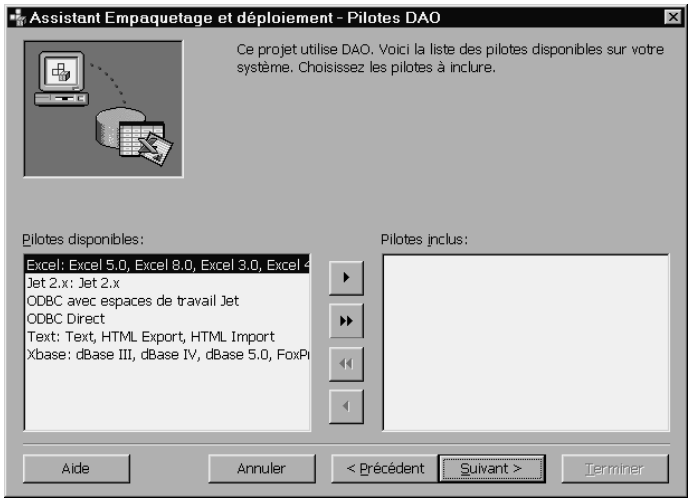


Figure 21.14
Vous pourrez avoir à sélectionner les fichiers d'accès aux données manuellement si votre application l'exige.

La boîte de dialogue peut différer selon le type d'accès aux données (ADO, DAO, etc.) utilisé par votre application.

Après que vous avez cliqué sur Suivant, l'assistant collecte tous les fichiers de l'application spécifiés et affiche la boîte de dialogue illustrée à la Figure 21.15. Vérifiez que tous les fichiers nécessaires à votre application y sont listés. Vous aurez éventuellement à ajouter (en cliquant sur Ajouter) d'autres fichiers, par exemple des fichiers LisezMoi.txt ou un fichier de base de données, pour que la routine d'installation les enregistre avec l'application dans le produit d'installation.

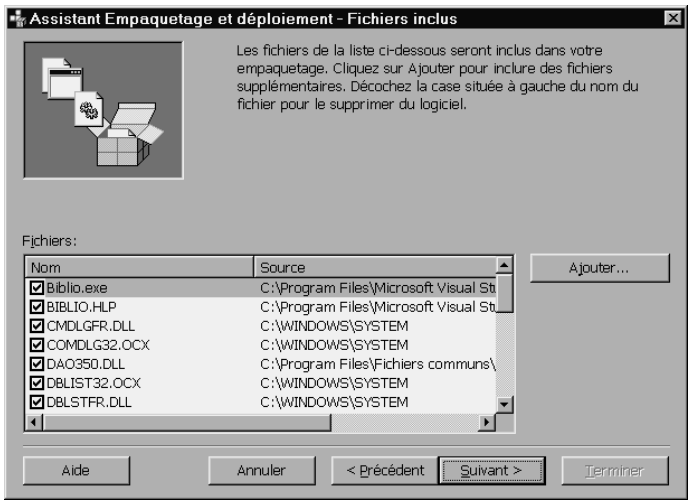


Figure 21.15
Parcourez les fichiers pour vous assurer que l'assistant a bien rassemblé tous les fichiers nécessaires au projet.

Soyez certains d'avoir les bons droits d'accès à tout contrôle ActiveX que vous distribuez avec votre application. Il se peut qu'une licence ne vous autorise pas à distribuer les contrôles ActiveX que vous n'avez pas créés. Consultez votre fournisseur de contrôles ActiveX pour connaître les règles de distribution.

La boîte de dialogue suivante demande des informations de distribution. Vous pouvez créer un fichier de distribution unique ou demander que la routine d'installation soit placée sur plusieurs disquettes ou d'autres types de support. Après avoir choisi le mode de découpage des fichiers d'installation, cliquez sur Suivant pour afficher le titre de l'écran d'installation. Saisissez le titre de votre projet, puis cliquez sur Suivant pour afficher la boîte de dialogue des icônes.

Dans cette boîte de dialogue, vous créez le sous-menu qui s'affichera dans le menu Démarrage de votre PC. En cliquant sur Nouvel élément, vous affichez la boîte de dialogue de la Figure 21.16, qui permet d'ajouter des éléments à l'entrée du menu Démarrage de cette application. Vous pouvez y ajouter un fichier LisezMoi ou un programme annexe (par exemple, un utilitaire système).

L'écran suivant détermine les chemins de destination de chacun des fichiers installés. Si la majorité des fichiers seront copiés dans le dossier sélectionné par l'utilisateur dans la procédure d'installation, ce que précise la variable système `AppPath`, vous pouvez sélectionner des fichiers particuliers dans la liste et les envoyer dans d'autres dossiers, par exemple le dossier Program Files (indiqué par la variable système `ProgramFiles`).

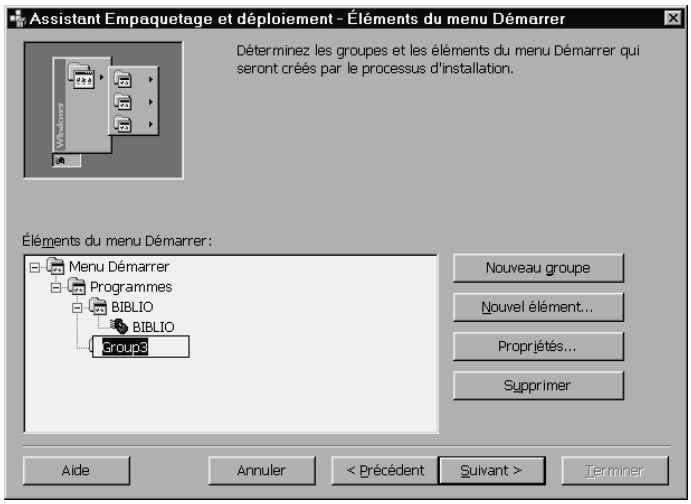


Figure 21.16
Vous pouvez déterminer la manière dont l'application se présentera dans le menu Démarrage de l'utilisateur.

Comme vous pouvez le voir, l'assistant Empaquetage et déploiement demande de prendre de nombreuses décisions. Elles permettent cependant d'avoir un contrôle complet de l'endroit et de la manière dont l'application arrive sur la machine de l'utilisateur.

Cliquez sur Suivant pour sélectionner tous les fichiers que vous voulez désigner comme partagés. Un fichier peut être partagé non seulement par les utilisateurs (comme un fichier de base de données auquel l'application a accès), mais aussi par d'autres programmes de l'ordinateur, comme ce peut être le cas de contrôles ActiveX contenus dans votre projet. Désignez les fichiers partagés en cochant la case correspondante.

Après avoir cliqué sur Suivant, vous verrez l'écran de fin de l'assistant qui vous demande le nom à donner au script d'installation. Créer un fichier script vous évitera d'avoir à répondre à la longue liste des demandes de l'assistant subie jusqu'ici la prochaine fois que vous créerez la routine d'installation. De plus, vous pouvez modifier ce script sans avoir à repasser par tous les écrans d'installation si quelque chose change dans le processus d'installation, par exemple la suppression d'un fichier partagé.

Une fois que vous cliquez sur Terminer, l'assistant crée le script d'installation, la routine d'installation, puis la place dans un ou plusieurs fichiers, suivant les options choisies. Quand tout est terminé, les fichiers d'installation sont sur votre machine, prêts à être distribués.

Après la génération de l'installation

Une fois que l'assistant a généré la routine d'installation, vous devez la tester, en exécutant le programme généré pour vous assurer qu'il n'y a pas de bogues et que l'application finale s'exécute sans problèmes.

Pour tester sérieusement la routine d'installation, exécutez le programme d'installation sur une machine qui n'a jamais contenu votre application. Mieux, assurez-vous qu'il n'y a pas même une copie de Visual Basic installée. Tester votre application sur ce type de machine nettoyée vous aide à garantir qu'elle s'installera correctement sur d'autres.

L'assistant Empaquetage et déploiement crée les fichiers d'installation à l'endroit spécifié lors du traitement. Vous y trouverez le fichier Setup.exe, le fichier Setup.lst (qui contient la liste de tous les fichiers liés à l'installation) et, éventuellement, d'autres fichiers dont l'extension se termine par le caractère souligné (_). Ces fichiers sont compressés ; la routine d'installation les décompresse sur la machine de destination.

La méthode la plus simple pour tester la routine d'installation générée consiste à choisir Exécuter à partir du menu Démarrage et de trouver le fichier du programme Setup.exe. Cliquez sur OK pour lancer l'installation. Une installation classique s'exécutera. Le programme analyse la machine de destination pour s'assurer qu'il n'y a pas de programme en exécution qui peut entrer en conflit avec un fichier à installer. La Figure 21.17 montre le dialogue d'avertissement affiché si l'utilisateur exécute le programme d'installation pendant que d'autres programmes utilisent certains des fichiers partagés de la routine.

Figure 21.17
Lancement du processus d'installation.

Si vous annulez le programme d'installation avant la fin de l'exécution, il supprime tous les fichiers copiés jusque-là, effaçant donc toute trace de l'installation de l'application.

Désinstaller l'application

L'assistant Empaquetage et déploiement ne se contente pas de créer la routine d'installation, il génère aussi une routine de désinstallation, connectée à l'icône Ajout/Suppression de programmes du Panneau de configuration, qui permet à l'utilisateur de désinstaller l'application à tout moment. Pour supprimer l'application du système, il suffit de suivre ces étapes :

1. A partir du menu Démarrage, sélectionnez Paramètres, puis Panneau de configuration.
2. Double-cliquez sur l'icône Ajout/Suppression de programmes.
3. Sélectionnez l'application dans la liste des applications installées. Après avoir vérifié que l'utilisateur souhaite supprimer l'application, la routine prend la main et supprime le programme et tous les fichiers qui lui sont liés.

Les informations de désinstallation sont enregistrées dans le même répertoire que l'application. Le fichier des instructions de suppression se nomme ST6UNST.LOG et il contient tous les détails nécessaires pour que l'utilitaire système Ajout/Suppression de programmes puisse faire son office. Tous les fichiers ne seront pas supprimés, en particulier les fichiers partagés par d'autres programmes. Avant de supprimer ces fichiers potentiellement nécessaires (comme les contrôles ActiveX), l'utilitaire de suppression affiche une boîte de dialogue d'avertissement qui laisse à l'utilisateur le soin de décider s'il faut ou non les supprimer.

En résumé

La leçon d'aujourd'hui a expliqué comment utiliser les puissants outils de débogage de Visual Basic. Apprendre comment déboguer une application devient rentable quand vous avez à pister des bogues. Si les outils de débogage de Visual Basic ne peuvent pas détecter les erreurs de logique, le débogueur simplifie leur recherche. Vous pouvez suivre à la trace l'exécution d'un programme, positionner des points d'arrêt et retrouver le déroulement de l'exécution d'un programme depuis le début.

Un des aspects les plus puissants du débogueur est son interaction avec le programme durant une session de points d'arrêts. Quand votre programme atteint un point d'arrêt, toutes les valeurs initialisées et calculées jusqu'à cet instant sont toujours actives. Vous pouvez donc consulter les variables pour voir si leurs valeurs intermédiaires sont celles

que vous attendez. Vous pouvez aussi changer la valeur d'une variable ou d'un contrôle au milieu de l'exécution du programme et en voir les conséquences sur la suite.

Une fois votre application déboguée, vous êtes prêt à la distribuer. La distribution des applications met en œuvre plus qu'une simple compilation. Vous devez prendre en compte la routine d'installation et vous assurer que les utilisateurs auront tous les fichiers liés à l'application.

Questions-réponses

Q Puis-je détecter toutes les erreurs de mon application lors de la rédaction du code si je coche la case Vérification automatique de la syntaxe ?

R Lorsque vous tapez votre programme, cette option cochée permettra certainement de détecter de nombreuses erreurs. Mais elle ne repère que les erreurs de syntaxe. Vous pouvez encore faire des erreurs d'exécution, comme des divisions par zéro. La fenêtre de code ne peut pas détecter ces erreurs.

Q Quelles sont les erreurs les plus simples à détecter ?

R Les erreurs de syntaxe sont les plus faciles à repérer. Non seulement elles sont faciles à repérer, mais en plus Visual Basic le fait à votre place. Si vous avez activé l'option Vérification automatique de la syntaxe, Visual Basic détectera toutes les erreurs à la rédaction ou, dans le cas contraire, à la première exécution. Les erreurs de logique sont les plus difficiles à repérer, car Visual Basic n'a aucune idée d'une anomalie. Vous-même ne réalisez pas toujours qu'il y a un problème au premier abord. Par exemple, si un total de balance générale se trompe d'un franc ou deux chaque jour, l'utilisateur peut ne pas remarquer l'erreur pendant un certain temps, ce qui risque entre-temps d'affecter d'autres données. C'est pourquoi il faut tester complètement son application.

Q Comment puis-je tester mon application avant de la distribuer ?

R La réponse dépend de la complexité de l'application. Si elle génère des détails cruciaux utilisés dans une prise de décision, comme un compte-rendu de paye ou de comptabilité, une des meilleures manières de la tester consiste à effectuer un *test en parallèle*. C'est-à-dire, exécuter votre application en parallèle au système actuel, que ce soit un système manuel ou une application informatique ancienne que vous remplacez. Comparez vos résultats au système en cours et faites faire la même chose à vos utilisateurs. Ce n'est qu'après avoir exécuté plusieurs cycles du système en parallèle (ce qui peut prendre un mois ou deux dans le cas d'une comptabilité), que vous aurez la confiance nécessaire pour remplacer le système en cours par votre application.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités, ainsi que des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se situent à l'Annexe A.

Quiz

1. Qu'est-ce qu'une erreur de syntaxe ?
2. Quelle est la différence entre une erreur de syntaxe et une erreur de logique ?
3. Vrai ou faux. Certaines erreurs à l'exécution peuvent entraîner l'arrêt de l'exécution du programme.
4. Vrai ou faux. Certaines erreurs de syntaxe peuvent entraîner l'arrêt de l'exécution du programme.
5. Comment pouvez-vous analyser les variables pendant l'exécution d'un programme ?
6. Comment le mode pas à pas peut-il vous aider à déboguer un programme ?
7. Vrai ou faux. Vous pouvez changer les valeurs des variables et des contrôles pendant l'exécution de l'application.
8. Quel est le plus rapide : un programme compilé ou un programme exécuté dans l'environnement Visual Basic ?
9. Quels sont les outils fournis par Visual Basic permettant de créer une routine d'installation de votre application ?
10. Que se passe-t-il si votre routine d'installation est plus volumineuse que le contenu d'une disquette, mais que vous devez la proposer sur ce support ?

Exercices

1. Quel type d'erreur contient l'instruction suivante ?

```
If (a < b) Therefore
   lblTitle.Caption = "Trop petit"
End If
```

2. Vrai ou faux. La phrase suivante contient deux types d'erreurs (cette question demande un peu de réflexion) :

   Cette phrase a deux erreurs.

Résumé de la Partie III

Félicitations, vous avez terminé votre formation de programmeur Visual Basic ! Vous comprenez désormais pratiquement tous les domaines du système Visual Basic. Vous avez les moyens de produire presque n'importe quel type d'application.

Tout ce dont vous avez maintenant besoin est d'acquérir de l'expérience.

Avec ce didacticiel et les Projets bonus, vous avez appris à quoi ressemblent les journées d'un programmeur Visual Basic. La programmation Visual Basic est très gratifiante. Naturellement, les emplois de programmeurs sont nombreux à pourvoir dans ce monde de haute technologie, mais la satisfaction tirée de la programmation ne saurait se limiter à une simple amélioration des revenus. En programmant avec Visual Basic, vous avez non seulement le pouvoir de créer tout type d'applications Windows dont vous pourriez avoir besoin, mais en outre, cela vous amusera. Et, plus important, Visual Basic n'oublie jamais ses racines ; il sera là pour vous aider à déboguer, ajouter du style aux programmes que vous écrivez et créer des exécutables compilés que vous pouvez distribuer pour permettre à d'autres de bénéficier du travail que vous avez effectué.

Dans cette partie, vous avez appris à maîtriser les points suivants :

- **Les modèles de feuilles.** Pourquoi réinventer la roue ? Utilisez les modèles de feuilles pour simplifier l'ajout de feuilles standard à vos applications (Chapitre 15).
- **Créer vos propres modèles de feuilles.** Si Visual Basic fournit plusieurs modèles de feuilles, vous pouvez aussi créer les vôtres et les ajouter à la collection proposée (Chapitre 15).
- **Modèles de feuilles de l'assistant Création d'applications.** L'assistant Création d'applications peut ajouter des modèles de feuilles à vos projets, mais vous devez toujours en modifier le code et les personnaliser pour qu'ils se conforment aux exigences de vos applications (Chapitre 15).

- **Contrôles OLE.** L'OLE (incorporation et liaisons d'objets) est un concept qui date presque des origines de Windows. D'autres outils, tels ActiveX, sont actuellement sous les feux de la rampe, mais de nombreux programmeurs travaillent encore avec des contrôles OLE pour incorporer des objets à leurs applications (Chapitre 16).
- **Objets.** Si certains puristes de la programmation sursautent en entendant dire que Visual Basic est un langage orienté objet, il supporte quand même de nombreuses constructions qui y sont apparentées (Chapitre 16).
- **Objets prédéfinis.** Visual Basic définit de nombreux objets, tels que `Screen`, `Printer` et `App`. Lors de l'exécution de votre application, vous pouvez utiliser ces objets prédéfinis pour en apprendre plus sur l'environnement de l'application (Chapitre 16).
- **Objets externes.** Vous avez un aperçu des exigences pour accéder aux objets qui sont en dehors de l'environnement Visual Basic, comme les valeurs d'une feuille de calcul Excel, à partir de votre application Visual Basic.
- **Collections d'objets.** Les collections d'objets vous permettent de travailler avec plusieurs éléments objets pris comme un groupe unique (Chapitre 16).
- **Contrôles ActiveX.** Un contrôle ActiveX peut fonctionner à de nombreux niveaux, d'un contrôle de la fenêtre Boîte à outils de Visual Basic à un programme d'une page Web (Chapitre 17).
- **Créer vos propres contrôles ActiveX.** Vous n'êtes pas obligé de vous limiter aux contrôles ActiveX fournis avec Visual Basic ou créés par d'autres. Vous pouvez écrire les vôtres et les utiliser dans vos projets (Chapitre 17).
- **Contrôle Data.** En ajoutant le contrôle Data à votre application, vous garantissez à vos utilisateurs non seulement l'accès à une base de données, mais aussi la possibilité de naviguer dans le fichier et d'y effectuer des modifications (Chapitre 18).
- **Programmer des jeux de données.** Après avoir appris quelques concepts simples, vous comprenez ce qui est nécessaire pour gérer une application de base de données complète sous une interface Visual Basic (Chapitre 18).
- **Accès Internet.** Ajoutez un navigateur Web dans vos applications (Chapitre 19).
- **Programmation Internet.** Contrôlez les traitements Internet dans votre application Visual Basic (Chapitre 19).
- **Ajouter de l'aide.** Donnez à vos utilisateurs les accès à l'aide dont ils ont besoin en installant une base de données d'aide complète dans votre application (Chapitre 20).
- **Options d'aide.** Visual Basic propose plusieurs moyens d'ajouter de l'aide à vos applications (Chapitre 20).

- **Aide par info-bulles.** A une petite échelle, les info-bulles peuvent être utiles pour l'utilisateur qui utilise les barres d'outils et les contrôles de votre application (Chapitre 20).
- **Déboguer le code.** Le débogueur interactif de Visual Basic simplifie la recherche et l'élimination des bogues du programme (Chapitre 21).
- **Compilation de l'application.** Avant de distribuer vos applications, il est préférable de compiler le code pour empêcher sa consultation ou sa modification, et pour permettre une exécution plus rapide de l'application (Chapitre 21).
- **Distribution du code.** Vous pouvez simplifier ou compliquer la vie de vos utilisateurs. Utilisez les outils de Visual Basic pour l'empaquetage de vos applications en une application distribuable, installable et blindée (Chapitre 21).

Chapitre 22

Tableaux multidimensionnels

Au Chapitre 10, nous avons déjà expliqué comment déclarer et utiliser les tableaux, qui contiennent des listes de valeurs. Si vous devez garder la trace de plusieurs éléments, vous pouvez les mettre dans des variables différentes ou les enregistrer dans un tableau sous un nom unique. L'avantage du tableau est de permettre de parcourir les variables à l'aide d'une instruction de boucle, telle que `For`.

Visual Basic supporte plusieurs types de tableaux. Vous pouvez déclarer un tableau de n'importe quel type de données pour contenir une liste de variables. Vous pouvez aussi déclarer un groupe de contrôles qui fonctionne comme un tableau de variables, et offre l'avantage de disposer de plusieurs objets de feuille sous le même nom et avec un ensemble de propriétés identiques. Chaque contrôle du groupe n'a cependant pas obligatoirement les mêmes valeurs de propriétés. Vous pouvez parcourir un groupe de contrôles aussi simplement qu'un tableau de variables lorsque vous voulez travailler sur les contrôles.

Cette leçon supplémentaire fait avancer d'un cran le concept de variables. Les tableaux étudiés jusqu'ici étaient des *tableaux à une dimension*, n'ayant qu'un indice. Dans cette leçon, vous allez apprendre comment étendre ce concept à la génération de tableaux, dits *tableaux multidimensionnels*. Ils sont souvent appelés *tables*, et permettent d'enregistrer les valeurs sous forme de lignes et de colonnes. Le nombre de dimensions peut dépasser deux pour créer des zones de stockage de données utiles.

Une fois que vous aurez appris les tableaux à plusieurs dimensions, vous étudierez le contrôle grille. Il contient des fonctionnalités permettant de présenter efficacement les données en deux dimensions ; le tableau à deux dimensions étant le plus souvent utilisé, le contrôle grille est alors pratique.

Vous apprendrez aujourd'hui :

- ce que sont les tableaux multidimensionnels ;
- comment déclarer les tableaux multidimensionnels ;
- les différentes manières d'initialiser les tableaux multidimensionnels ;
- comment les boucles aident au traitement des tableaux multidimensionnels ;
- les limites que pose Visual Basic sur les tableaux multidimensionnels ;
- le contrôle grille ;
- comment les propriétés du contrôle grille déterminent le nombre de lignes et de colonnes de la grille finale ;
- les méthodes utilisées pour assigner les données aux grilles ;
- la propriété `FormString` ;
- comment afficher des images dans les cellules de grille.

Introduction aux tableaux multidimensionnels

Certaines données s'adaptent à un tableau à une dimension (que vous avez étudié jusqu'ici), d'autres à un format de table. Les tableaux à une dimension n'ont qu'un indice. Ils représentent une liste de valeurs. La Figure 22.1 montre qu'un tableau à une dimension a une notion de longueur et de direction, exactement comme une ligne.

N'oubliez pas que les tableaux sont stockés dans la mémoire de votre ordinateur sans aucune notion de la direction, illustrée à la Figure 22.1. L'idée de direction unique fonctionne cependant bien pour démontrer la nature linéaire d'un groupe d'éléments de tableaux. Si vous ajoutez une dimension, vous ajoutez aussi une idée de direction supplémentaire, comme vous le verrez dans un moment lorsque nous parlerons des tableaux à deux dimensions.

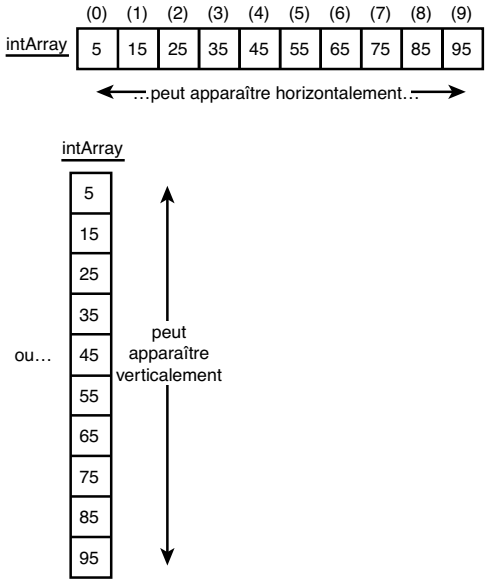


Figure 22.1
Un tableau à une dimension a une longueur et une direction.

Le reste de cette leçon explique comment utiliser les tableaux à plus d'une dimension, dits *tableaux multidimensionnels*. Ils sont aussi nommés *tables* ou *matrices* et se composent de lignes et de colonnes.

Un tableau multidimensionnel a plus d'un indice et, par là même, plus d'une dimension. Le nombre de dimensions dépend du nombre de directions. Un tableau à une dimension a une direction, un tableau à trois dimensions a trois directions, comme un objet dans l'espace a une largeur, une longueur et une hauteur.

Supposons qu'une équipe de handball souhaite suivre les buts marqués par ses joueurs. L'équipe a joué huit matches, et il y a dix joueurs dans l'équipe. Le Tableau 22.1 montre l'enregistrement des buts de l'équipe.

Remarquez-vous que ce tableau a deux dimensions ? Il a des lignes (première dimension) et des colonnes (deuxième dimension). On parle donc d'une table à deux dimensions de dix lignes et huit colonnes. (On désigne, en général, les lignes en premier.) Une table à deux dimensions a donc deux directions — horizontale et verticale. Comme les tableaux à une dimension, l'enregistrement en mémoire des tableaux à deux dimensions n'est pas littéralement organisé de cette manière, mais le langage Visual Basic permet de manipuler les données comme si elles étaient enregistrées en lignes et en colonnes.

Tableau 22.1 : Le tableau des buts d'une équipe de handball fonctionne bien sous forme de table

Joueur	*Partie 1*	*Partie 2*	*Partie 3*	*Partie 4*	*Partie 5*	*Partie 6*	*Partie 7*	*Partie 8*
Adams	2	1	0	0	2	3	3	1
Berryhill	1	0	3	2	5	1	2	2
Edwards	0	3	6	4	6	4	5	3
Grady	1	3	2	0	1	5	2	1
Howard	3	1	1	1	2	0	1	0
Powers	2	2	3	1	0	2	1	3
Smith	1	1	2	1	3	4	1	0
Townsend	0	0	0	0	0	0	1	0
Ulmer	2	2	1	1	2	1	1	2
Williams	2	3	1	0	1	2	1	1

Une matrice, comme un tableau à une dimension, n'a qu'un nom. De plus, une matrice ne peut contenir qu'un seul type de données — le type de sa déclaration. Si vous déclarez un type Variant pour la matrice, les cellules peuvent contenir tous les types de données qui peuvent être représentées par le type Variant.

Une cellule est un élément d'un tableau à plusieurs dimensions. Dans un tableau à une dimension, une cellule est un élément de la liste, mais dans un tableau à deux dimensions (comme la liste des buts de l'équipe de handball du Tableau 22.1), une cellule se compose de l'intersection d'une ligne et d'une colonne.

Dans le Tableau 22.1, chaque ligne correspond au nom du joueur et à chaque colonne est associée une partie, mais ces en-têtes ne font pas partie du tableau. Les données ne comprennent que quatre-vingts valeurs (dix lignes par huit colonnes). Dans ce cas, ce sont des entiers. Si le tableau se compose de noms, c'est un tableau de chaînes de caractères, etc.

Le nombre de dimensions — ici, deux — correspond aux dimensions du monde physique. La première représente une ligne : un tableau à une dimension représente une ligne, ou une liste, de valeurs. Deux dimensions représentent la longueur et la largeur. Vous écrivez sur une feuille de papier en deux dimensions — c'est la représentation d'une surface plane. Trois dimensions représentent la largeur, la longueur et la hauteur. Vous pouvez avoir vu des films en 3D : les images ont non seulement une largeur et une hauteur, mais elles donnent l'impression d'avoir aussi une profondeur (impression qui prend corps lorsqu'on regarde tout simplement autour de soi).

La raison d'une telle insistance sur la manière dont les gens voient les dimensions est que votre travail de programmeur est simplifié si vous visualisez mentalement les données multidimensionnelles dans l'espace avec leurs lignes et leurs colonnes (et éventuellement plus).

Bien que Visual Basic vous permette de travailler jusqu'à soixante dimensions, il est difficile d'en visualiser plus de trois. Vous pouvez cependant penser à chaque dimension qui suit la troisième comme une nouvelle occurrence. En d'autres termes, vous pouvez enregistrer une liste des buts d'un joueur sur la saison dans un tableau. Les scores de l'équipe (voir Tableau 22.1) sont en deux dimensions. La ligue, composée des enregistrements de plusieurs équipes, représente un tableau à trois dimensions. Chaque équipe (la profondeur du tableau) a des lignes et des colonnes de données sur les buts. S'il y a plus d'une ligne, elles peuvent représenter une autre dimension.

Malgré la capacité généreuse de soixante dimensions proposée par Visual Basic, vous écrirez rarement des programmes qui demandent plus de trois ou peut-être quatre dimensions. La plupart du temps, vous travaillerez sur des tableaux à une ou deux dimensions.

Déclarer les tableaux multidimensionnels

Comme pour les tableaux à une dimension, vous utiliserez les instructions `Dim` ou `Public` pour réserver la place des tableaux multidimensionnels. Au lieu de mettre une valeur entre parenthèses, vous en mettrez une par dimension. Les formats de base pour la réservation des tableaux multidimensionnels sont les suivants :

```
Public taName(intSub) [As dataType][, taName(intSub) [As dataType]]...
Dim taName (intSub) [As dataType][, taName (intSub) [As dataType]]...
```

Les valeurs `intSub` du tableau peuvent prendre ce format général :

```
[intLow To] intHighRow[, [intLow To] intHighColumn][, [intLow To]
➥intHighDepth][,...]
```

Comme c'est le cas des tableaux à une dimension, la réservation effective est plus simple que ce qu'en laisse penser le format. Pour déclarer les données de l'équipe du Tableau 22.1, par exemple, vous pouvez utiliser l'instruction `Dim` suivante :

```
Dim intTeams(1 To 10, 1 To 8) As Integer
```

Cette instruction réserve en mémoire un tableau à deux dimensions de quatre-vingts éléments. Les indices des éléments sont illustrés à la Figure 22.2.

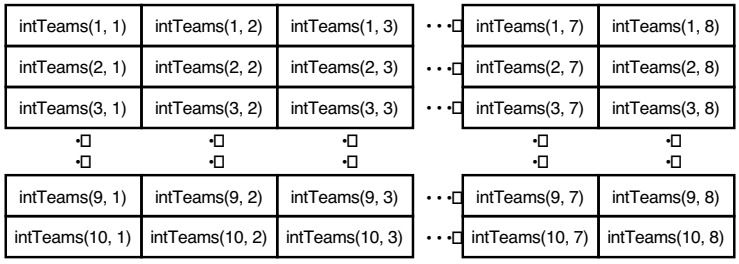


Figure 22.2
Le tableau de l'équipe de handball demande deux ensembles d'indices.

Si vous avez à suivre une ligue de quinze équipes, vous ajouterez un autre indice :

```
Dim intTeams(1 To 15, 1 To 10, 1 To 8) As Integer
```

Le premier indice indique chaque équipe, le second, le nombre de joueurs de chaque équipe, et le dernier le nombre de parties.

Pensez à un tableau à trois dimensions comme à un échiquier à trois dimensions, avec des échiquiers empilés les uns sur les autres. Une représentation à quatre dimensions serait plusieurs ensembles d'échiquiers. La quatrième dimension serait le numéro correspondant à chaque ensemble 3D d'échiquiers.

Comment connaître l'ordre des indices, et savoir si l'indice de droite représente des colonnes ? Vous ne pouvez pas le savoir, car les indices peuvent représenter ce que vous voulez. Cependant, la norme pour un tableau à deux dimensions est de considérer l'indice de gauche comme la ligne et l'indice de droite comme la colonne. En passant de deux dimensions à trois, l'indice ajouté est presque toujours le premier pour garantir que les deux derniers représentent les lignes et les colonnes d'une table. En gardant les indices présentant la table comme les deux derniers, vous aidez à leur cohérence.

L'instruction suivante réserve assez de mémoire pour les programmes d'une chaîne de télévision sur une semaine :

```
Dim strShows(1 To 7, 1 To 48) As String
```

Cette instruction réserve sept jours (les lignes) de programmes de trente minutes (les jours faisant vingt-quatre heures, la table contient quarante-huit colonnes).

Comme vous le savez, tous les éléments d'une table doivent avoir le même type de données. Ici, c'est une variable String.

Vous pouvez initialiser certains éléments avec les instructions suivantes :

```
strShows(3, 12) = "Quand l'hôpital tourne"
strShows(1, 5) = "Le jeu des devinettes"
strShows(7, 20) = "Publireportage sur le Thé glacé aux framboises"
```

La réservation de l'espace de plusieurs tableaux multidimensionnels consomme vite de la mémoire. Les instructions suivantes réservent une grande quantité de mémoire :

```
Public ara1(10, 20) As Single
Dim ara2(4, 5, 5) As Double
Public ara3(6, 10, 20, 30) As Integer
```

`ara1` occupe deux cents emplacements mémoire de variables simple précision, `ara2` occupe cent emplacements double précision, et `ara3` occupe trente-six mille emplacements mémoire. Comme vous pouvez le voir, le nombre d'éléments augmente vite. Soyez attentif à ne pas réserver trop d'emplacements de tableaux pour ne pas épuiser la quantité de mémoire disponible.

En lisant des données tabulaires dans des tableaux multidimensionnels et en travaillant sur les données d'un tableau à la place des tables de bases de données, vous pouvez accélérer les temps d'exécution du programme. Tout ce qui se passe en mémoire est plus rapide que la lecture et l'écriture disque à chaque accès aux valeurs. Cependant, l'espace mémoire est plus réduit que l'espace disque. Si vous travaillez sur des fichiers importants, vous devez renoncer à l'efficacité de la mémoire pour la capacité du disque.

Les tableaux et les boucles *For*

Comme vous le verrez dans certains des exemples de programmes suivants, les boucles `For` imbriquées sont de bonnes candidates pour le parcours de tous les éléments d'un tableau multidimensionnel. Par exemple, le Listing 22.1 imprime les six valeurs d'indices possibles d'un tableau multidimensionnel dans des boîtes de message successives.

Listing 22.1 : Les boucles imbriquées permettent de parcourir rapidement les tables

```
1:  For intRow = 1 To 2
2:    For intCol = 1 To 3
3:      MsgBox("Ligne : " & intRow & ", Colonne : " & intCol)
4:    Next intCol
5:  Next intRow
```

Si vous exécutez le code du Listing 22.1, vous verrez les sorties suivantes dans les boîtes de message :

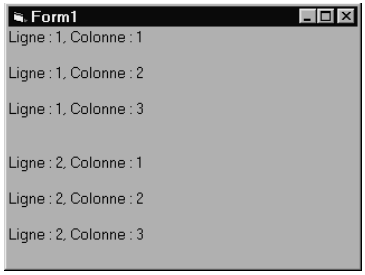
```
Ligne : 1, Colonne : 1
Ligne : 1, Colonne : 2
Ligne : 1, Colonne : 3
Ligne : 2, Colonne : 1
Ligne : 2, Colonne : 2
Ligne : 2, Colonne : 3
```

Vous pouvez modifier le code et utiliser des méthodes `Print` pour imprimer directement sur la feuille à la place de boîtes de message. `Print` est sans nul doute une bonne méthode à utiliser pour pratiquer la manipulation des indices de tableaux multidimensionnels, car elle permet de placer simplement de grandes quantités de sorties sur une feuille. Par exemple, le Listing 22.2 utilise la méthode `Print` pour imprimer directement sur une feuille. La Figure 22.3 montre ce à quoi pourrait ressembler la feuille.

Listing 22.2 : Les boucles imbriquées fournissent des indices pour parcourir tout le tableau

```
1:  For intRow = 1 To 2
2:    For intCol = 1 To 3
3:      Form1.Print "Row: " & intRow & ", Col: " & intCol
4:    Next intCol
5:    Form1.Print
6:  Next intRow
```

Figure 22.3
Vous pouvez expérimenter l'impression des valeurs de tableaux avec Print pour consulter toutes les sorties sur une même feuille.



Attention aux résultats de la Figure 22.3. N'oubliez pas que l'objectif du Listing est de vous montrer comment des boucles `For` imbriquées permettent de parcourir une table par lignes et par colonnes. Les valeurs imprimées de la Figure 22.3 ne sont pas des valeurs de tableau, mais les indices d'un tableau déclaré de deux lignes et de trois colonnes.

Si vous avez à imprimer les indices, par ordre de ligne, d'un tableau de deux lignes et trois colonnes dimensionné par l'instruction `Dim` suivante, vous verriez les numéros d'indices illustrés par les boucles imbriquées du Listing 22.2.

```
Dim intTable(1 To 2, 1 To 3)
```

Remarquez qu'il y a autant d'instructions `For…Next` que d'indices dans l'instruction `Dim` (deux !). La boucle extérieure représente le premier indice (lignes) et la boucle intérieure, le second (colonnes). Les boucles imbriquées sont sans nul doute la manière la plus courante de parcourir les éléments d'un tableau ; leur maîtrise est donc cruciale pour programmer efficacement avec des tableaux multidimensionnels.

Initialiser les tableaux

Vous pouvez initialiser les éléments d'un tableau multidimensionnel de plusieurs façons, dont :

- assigner des valeurs aux éléments de la table ;
- utiliser `InputBox` pour remplir les éléments, un par un, à partir d'une boîte de message ;
- lire les valeurs, une par une, à partir d'un disque ou d'un fichier de base de données ;
- calculer les valeurs à partir d'autres valeurs.

En réfléchissant à cette liste, vous constaterez qu'on initialise les tables et tous les tableaux multidimensionnels comme on le fait avec toute autre variable. Cette méthode, cependant, vous permet de penser aux données sous une forme tabulaire, ce qui aide à accélérer la programmation et la maintenance.

La plupart des données de tableaux multidimensionnels proviennent de données de feuilles ou — plus souvent — de fichiers disques. Quelle qu'en soit la provenance, les boucles `For` imbriquées sont d'excellentes instructions de commande du parcours des indices. L'exemple suivant illustre un peu mieux comment elles peuvent fonctionner avec des tableaux multidimensionnels.

Supposons qu'une entreprise informatique vende deux tailles de disquettes : 3 1/2 et 5 1/4 pouces. Chaque disquette existe en quatre capacités : simple face, basse densité ; double face, basse densité ; simple face, haute densité ; double face, haute densité. Cet inventaire des disquettes se prête bien à un tableau à deux dimensions. Les prix de détail des disquettes sont les suivants :

	Simple face Basse densité	*Double face Basse densité*	*Simple face Haute densité*	*Double face Haute densité*
3 1/2	2,30	2,75	3,20	3,50
5 1/4	1,75	2,10	2,60	2,95

La procédure du Listing 22.3 enregistre le prix de chaque disque dans une table et imprime les valeurs sur la feuille en utilisant des boucles `For` imbriquées. Vous pouvez insérer cette procédure dans un module standard ou une procédure événementielle pour en déclencher l'exécution.

Listing 22.3 : Les éléments d'inventaire apparaissent souvent dans une table

```
 1:  Private Sub disks ()
 2:  ' Assigne et imprime les prix des disquettes
 3:    Dim curDisks(1 To 2, 1 To 4) As Currency
 4:    Dim intRow As Integer, intCol As Integer
 5:    ' Assigne le prix de chaque élément
 6:    curDisks(1, 1) = 2.30    ' Ligne 1, colonne 1
 7:    curDisks(1, 2) = 2.75    ' Ligne 1, colonne 2
 8:    curDisks(1, 3) = 3.20    ' Ligne 1, colonne 3
 9:    curDisks(1, 4) = 3.50    ' Ligne 1, colonne 4
10:   curDisks(2, 1) = 1.75    ' Ligne 2, colonne 1
11:   curDisks(2, 2) = 2.10    ' Ligne 2, colonne 2
12:   curDisks(2, 3) = 2.60    ' Ligne 2, colonne 3
13:   curDisks(2, 4) = 2.95    ' Ligne 2, colonne 4
14:   ' Imprime les prix sous forme de table
15:   Form1.Print
16:   Form1.Print Tab(12); "Simple face,    Double face,      ";
17:   Form1.Print "Simple face,    Double face"
18:   Form1.Print Tab(12); "faible densité   faible densité    ";
19:   Form1.Print "haute densité   haute densité"
20:   For intRow = 1 To 2
21:     If (intRow = 1) Then
22:        Form1.Print "3 pouces 1/2 "; Tab(15);
23:     Else
24:        Form1.Print "5 pouces 1/4"; Tab(15);
25:     End If
26:     For intCol = 1 To 4
27:       Form1.Print curDisks(intRow, intCol); Spc(8);
28:
29:     Next intCol
30:     Form1.Print   ' Déplace le curseur à la ligne suivante
31:   Next intRow
32: End Sub
```

Cette procédure génère la sortie illustrée à la Figure 22.4, une fois la fenêtre redimensionnée pour afficher toute la table. Bien que réduit, le tableau multidimensionnel de 2 par 4 montre comment vos données peuvent parfois bien s'adapter à un enregistrement en tableau. Le code de l'exemple peut paraître assez long pour une table qui ne comprend que huit valeurs. N'oubliez cependant pas que vous n'initialisez que rarement les tableaux comme le font les lignes 6 à 13. Pour ce petit exemple, et comme introduction aux tableaux, les assignations sont peut-être la meilleure manière de commencer.

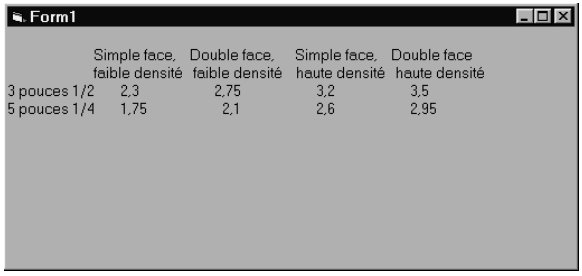


Figure 22.4
Le prix des disquettes s'affiche sous forme tabulaire.

Dans le Chapitre 10, vous avez étudié la fonction `Array()`*, qui assigne un ensemble de valeurs à tout un tableau en une seule instruction. N'utilisez pas la fonction* `Array()` *dans les tableaux multidimensionnels, car elle ne fonctionne qu'avec les tableaux à une dimension.*

Le contrôle grille

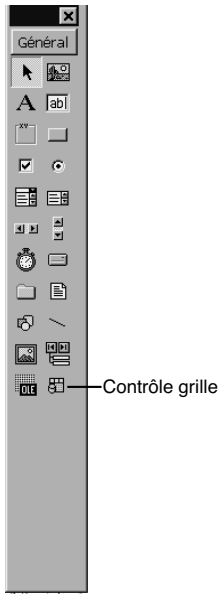
Le tableau à deux dimensions le plus courant, la table, se présente au mieux sous forme de lignes et de colonnes. Le contrôle grille propose une manière pratique d'afficher les tables. Les utilisateurs peuvent naviguer dans les valeurs en utilisant des barres de défilement, qui sont automatiquement affichées si le contrôle n'est pas aussi grand que la table.

Préparer le contrôle grille

Le contrôle grille ne fait pas partie de la boîte à outils standard. Il faut suivre ces étapes pour l'ajouter :

1. Appuyez sur Ctrl-T pour ouvrir la boîte de dialogues Composants.
2. Sélectionnez Microsoft FlexGrid Control 6.0.
3. Cliquez sur OK. Le contrôle MSFlexGrid (désigné habituellement comme le contrôle grille) s'affiche dans la fenêtre Boîte à outils (voir Figure 22.5).

Figure 22.5
Le contrôle grille permet d'afficher des données tabulaires.



Info

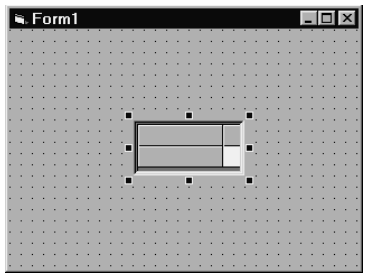
Visual Basic comprend plusieurs types de contrôles grille. Vous les découvrirez en balayant la liste de la boîte de dialogue Composants. Certains sont des contrôles liés qui, comme vous l'avez appris au dix-huitième chapitre, sont liés aux bases de données pour que l'utilisateur puisse consulter les données sous-jacentes, affichées dans la table. Le contrôle de grille lié ne permet que la lecture des données d'une base. Selon la version (Entreprise, Professionnelle ou Standard), vous trouverez un ensemble différent de grilles lorsque vous afficherez la liste des Composants.

Comprendre comment fonctionne le contrôle grille

Quand vous placez le contrôle grille sur une feuille, vous devez le redimensionner avant qu'il ne prenne une apparence tabulaire. Lorsque vous en augmentez la taille, il ne ressemble pas trop à une table (voir Figure 22.6). Le problème est que le nombre de lignes et de colonnes par défaut est de deux. Lorsque vous ajouterez des lignes et des colonnes, le contrôle grille ressemblera plus à une table de ce nom.

Une des premières tâches à effectuer après avoir ajouté le contrôle grille est d'augmenter le nombre de lignes et de colonnes jusqu'à ce que la grille en contienne un nombre raisonnable. Il n'est pas obligatoire d'afficher l'intégralité des lignes et des colonnes du tableau multidimensionnel, car des barres de défilement permettront à l'utilisateur de naviguer dans les données.

Figure 22.6
Le contrôle grille ne ressemble tout d'abord pas vraiment à une table lorsque vous le placez sur une feuille.

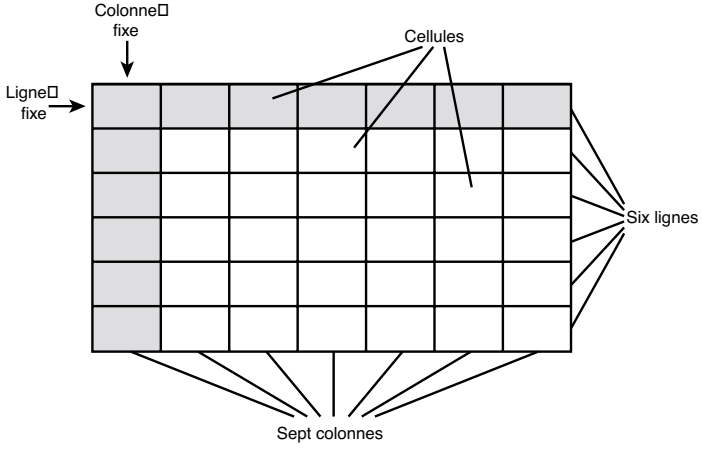


Le contrôle grille supporte des lignes et des colonnes fixes (voir Figure 22.7). Vous pouvez commander leur nombre. Si vous augmentez le nombre de lignes ou de colonnes fixes, la zone grisée augmente d'autant.

Lignes et colonnes fixes indiquent les lignes et les colonnes d'un contrôle grille qui ne défilent pas lorsque l'utilisateur clique sur les barres de défilement. Elles permettent d'indiquer des labels décrivant les données de la grille, un peu comme les numéros et les noms de colonnes et de lignes pour une cellule de feuille de calcul.

Les lignes et les colonnes fixes sont souvent désignées comme les en-têtes de lignes et de colonnes.

Figure 22.7
Les lignes et les colonnes fixes offrent une place pour les labels.



Une fois le contrôle de grille de données rempli, les utilisateurs peuvent faire défiler les données et sélectionner une ou plusieurs cellules de la grille. Par programmation, vous pouvez mettre à jour ou copier les valeurs des cellules sélectionnées depuis ou vers d'autres cellules, variables ou contrôles.

Le contrôle grille contient toutes sortes d'informations : texte, nombres ou même images.

Le Tableau 22.2 contient plusieurs propriétés uniques ou importantes du contrôle grille. Il ne décrit pas les propriétés déjà connues, telles que `Height` ou `Width`, ni d'autres plus obscures. En les étudiant, vous en apprendrez beaucoup sur la manière d'utiliser ce contrôle. Nombre d'entre elles se rapportent à la sélection des cellules dans les données tabulaires de la grille.

Tableau 22.2 : Le contrôle grille supporte plusieurs propriétés importantes lors de la conception

Propriété	*Description*
`AllowBigSelection`	Permet à l'utilisateur de sélectionner une ligne ou une colonne entière par un simple clic de souris. Positionnée à `True`, cette propriété lui permet aussi de cliquer dans l'en-tête pour sélectionner la totalité de la ligne ou de la colonne.
`AllowUserResizing`	Détermine la liberté donnée à l'utilisateur de redimensionner la largeur et la hauteur des colonnes et des lignes. Si cette propriété est paramétrée à `flexResizeNone`, l'utilisateur ne peut pas faire de modifications. Si elle est à `flexResizeColumns`, il peut changer la largeur des colonnes. Si elle est à `flexResizeRows`, il peut changer la hauteur de ligne. Si elle est paramétrée à `flexResizeBoth`, il peut changer la largeur et la hauteur. Selon les données, l'utilisateur peut avoir besoin de redimensionner les lignes et les colonnes pour pouvoir les lire dans le contrôle.
`Cols`	Définit le nombre de colonnes du contrôle grille.
`(Personnalisé)`	Ouvre la boîte de dialogue Pages de propriétés du contrôle grille (voir Figure 22.8). Elle permet de paramétrer facilement plusieurs propriétés.
`FillStyle`	Spécifie le format d'une cellule ou d'un ensemble de cellules. Si elle est configurée à `flexFillSingle`, la cellule en cours est formatée. Si elle est à `flexFillRepeat`, toutes les cellules sélectionnées seront formatées.
`FixedCols`	Indique le nombre de colonnes fixes (qui ne défilent pas) utilisées pour les en-têtes.

Tableau 22.2 : Le contrôle grille supporte plusieurs propriétés importantes lors de la conception *(suite)*

Propriété	*Description*
`FixedRows`	Indique le nombre de lignes fixes (qui ne défilent pas) utilisées pour les en-têtes.
`FocusRect`	Détermine comment s'affiche la sélection par l'utilisateur de la cellule courante. Si cette propriété est paramétrée à `flexFocusNone`, la cellule active n'a pas de surbrillance particulière. Si elle a la valeur `flexFocusLight`, un contour léger entoure la cellule sélectionnée. Si elle est à `flexFocusHeavy`, la cellule en cours est affichée en surbrillance bleue.
 *Si vous paramétrez la propriété `FocusRect` à `flexFocusHeavy`, la cellule ne se détachera pas dans un groupe de cellules sélectionnées.*	
`FormatString`	Contient une chaîne de caractères qui détermine la manière dont une cellule ou un groupe de cellules doivent être formatées. Voyez la section sur la propriété `FormatString`, plus loin dans ce chapitre.
`GridLines`	Précise comment doivent se présenter les lignes non fixes de la grille. Si cette propriété est paramétrée à `flexGridNone`, il n'y a pas de séparation des cellules. Si elle est à `flexGridFlat`, des lignes grises séparent les cellules les unes des autres. Si elle est paramétrée à `flexGridInset`, des lignes sombres séparent les cellules. `flexGridRaised` montre les cellules soulevées dans un effet 3D.
`GridLinesFixed`	Précise l'apparence des lignes fixes de la grille. Si cette propriété est paramétrée à `flexGridNone`, il n'y a pas de séparation des cellules. Si elle est à `flexGridFlat`, des lignes grises séparent les cellules les unes des autres. Si elle est paramétrée à `flexGridInset`, des lignes sombres séparent les cellules. `flexGridRaised` montre les cellules soulevées dans un effet 3D.
`HighLight`	Détermine comment s'affichent les cellules sélectionnées. Si cette propriété est paramétrée à `flexHighlightNever`, la sélection des cellules n'apparaît jamais à l'écran. Si elle est à `flexHighlightAlways`, les cellules sélectionnées apparaissent en surbrillance (l'arrière-plan est assombri). Si elle est paramétrée à `flexHighlightWithFocus`, les cellules sélectionnées n'apparaissent que lorsque le contrôle est actif.
`RowHeightMin`	Spécifie le nombre de twips minimal d'une ligne, interdisant à l'utilisateur de la réduire plus.

Tableau 22.2 : Le contrôle grille supporte plusieurs propriétés importantes lors de la conception *(suite)*

Propriété	*Description*
`Rows`	Spécifie le nombre de lignes qui s'afficheront dans le contrôle grille.
`SelectionMode`	Détermine comment se produit la sélection des cellules. Si cette propriété est paramétrée à `flexSelectionFree`, l'utilisateur peut sélectionner n'importe quel ensemble rectangulaire de cellules. Si elle est à `flexSelectionByRow`, les cellules sont sélectionnées par lignes entières. Si elle est paramétrée à `flexSelectionByColumn`, la sélection des cellules se fait par colonnes entières.
`WordWrap`	Définit si le contenu des cellules peut s'adapter à la modification de la largeur ou de la hauteur par l'utilisateur.

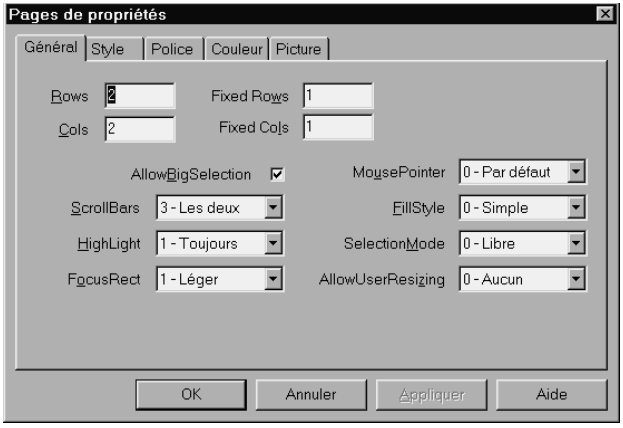


Figure 22.8
La boîte de dialogue Pages de propriétés permet de paramétrer facilement les propriétés courantes du contrôle grille.

Le contrôle grille a la particularité de comprendre plusieurs propriétés d'exécution qui doivent obligatoirement être paramétrées pour qu'il puisse fonctionner. Le Tableau 22.3 contient de nombreuses propriétés d'exécution, dont certaines doivent être paramétrées dans votre programme.

Certaines propriétés de formatage, comme `ColAlignment`, sont là pour vous permettre de contrôler le format à l'exécution. Pour formater une cellule lors de la conception, utilisez la propriété `FormatString` du Tableau 22.3.

Tableau 22.3 : Votre code Visual Basic doit paramétrer plusieurs propriétés à l'exécution

Propriété	*Description*
`CellAlignment`	Définit l'alignement des valeurs à l'intérieur des cellules. Si cette propriété est paramétrée à `flexAlignmentTop (0)`, le contenu des cellules est aligné sur le coin supérieur gauche. Si elle est à `flexAlignCenter (1)`, le contenu des cellules est aligné à gauche et centré à la verticale. Les autres valeurs de constantes nommées qui alignent le contenu des cellules de différentes manières sont : `flexAlignLeftBottom (2)`, `flexAlignCenterTop (3)`, `flexAlignCenterCenter (4)`, `flexAlignCenterBottom (5)`, `flexAlignRightTop (6)`, `flexAlignRightCenter (7)`, `flexAlignRightBottom (8)`, et `flexAlignGeneral (9)` est le comportement par défaut avec des chaînes alignées à gauche et des nombres alignés à droite.
`Col`	Paramètre le numéro de colonne de la cellule dont vous voulez changer la valeur. Une cellule est repérée en partant de l'intersection de ligne et de colonne 0,0 dans le coin supérieur gauche. La valeur donnée à la propriété `Text` du contrôle de grille va dans la cellule définie par la valeur de `Col`.
`ColAlignment`	Détermine l'alignement des valeurs dans une colonne particulière en utilisant les mêmes constantes nommées que pour `CellAlignment`.
`ColWidth`	Détermine la largeur en twips d'une colonne.
`Row`	Configure le numéro de ligne de la cellule dont vous voulez changer la valeur. Une cellule est repérée en partant de l'intersection de ligne et de colonne 0,0 dans le coin supérieur gauche. La valeur donnée à la propriété `Text` du contrôle de grille va dans la cellule définie par la valeur de `Row`.
`SelEndCol`	Spécifie le numéro de la dernière colonne de droite d'une sélection.
`SelEndRow`	Spécifie le numéro de la ligne la plus basse de la sélection.
`SelStartCol`	Spécifie le numéro de la colonne la plus à gauche de la sélection.
`Text`	Spécifie le contenu de la cellule donnée. Vous pouvez assigner à la cellule en cours (sélectionnée par l'intersection des propriétés `Row` et `Col`) une nouvelle valeur en l'attribuant à la propriété `Text`.

De nombreuses autres propriétés de conception et d'exécution existent, mais les Tableaux 22.2 et 22.3 listent les plus importantes pour un utilisateur qui découvre le contrôle grille.

Le contenu des cellules est paramétré ou lu par la propriété `Text`, mais cela ne veut pas dire que les données doivent être des chaînes de caractères. Vous pouvez assigner des valeurs numériques à la propriété `Text` ; Visual Basic les convertira en chaînes avant de faire l'assignation. De même, vous pouvez assigner une cellule à une variable numérique. Visual Basic convertira le texte dans la cellule en nombre avant de faire l'assignation. Si la valeur ne peut pas être convertie en nombre par Visual Basic, une erreur se produira.

Utiliser le contrôle grille dans une application

Une des meilleures manières de comprendre le contrôle grille consiste à créer une application qui l'utilise. Les quelques sections suivantes créent une application incluant des tableaux multidimensionnels et un contrôle grille. L'application utilise ce dernier pour gérer des cellules sélectionnées afin que vous voyiez comment travailler sur une plage de cellules.

Configurer le projet

La Figure 22.9 montre l'écran correspondant à l'application que vous allez créer. Le contrôle grille affiche les commissions sur les ventes pour des produits et des personnels de vente particuliers. Le nom des vendeurs s'affiche en en-tête de colonne et les noms de produits sur la gauche de la grille..

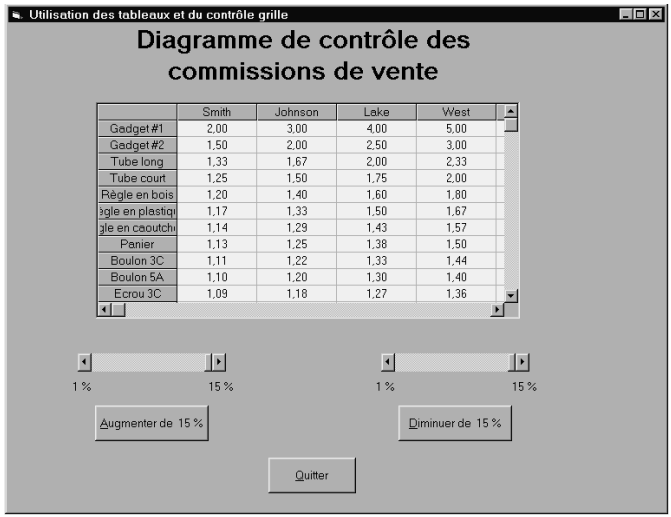


Figure 22.9
Le contrôle grille permet une gestion facile des commissions de vente.

Toutes les données sont stockées dans un tableau à deux dimensions et transférées dans le contrôle grille. Cette application ne charge pas le tableau initial à partir du disque et ne l'enregistre pas à la sortie. L'origine des valeurs du tableau est une série d'instructions d'assignations. Dans une véritable application, vous chargerez et enregistrerez le tableau sur disque, probablement dans une base de données. Si cette application comprenait toutes les entrées/sorties disque, le projet serait trop étendu pour se concentrer efficacement sur les tables et le contrôle grille

Les points suivants mettent en évidence l'utilité de cette application :

- Les barres de défilement peuvent être utilisées pour augmenter ou diminuer les revenus des ventes de tout vendeur sur n'importe quel produit.
- Le montant d'augmentation ou de diminution peut être modifié par les barres de défilement, ce qui change dynamiquement le titre des boutons.
- La cellule en cours est la seule qui change quand l'utilisateur augmente ou diminue le pourcentage de commission, à moins qu'il ait sélectionné une plage de cellules, auquel cas le pourcentage sera modifié sur toute la plage.
- La plupart des valeurs par défaut des propriétés du contrôle grille fonctionneront dans ce projet. Comme d'habitude, Microsoft a choisi les valeurs de propriétés les plus courantes comme valeurs par défaut. Vous aurez à paramétrer des valeurs de propriétés à l'exécution (comme le décrit la section précédente), mais à la conception, il suffira de modifier le nom, le nombre de colonnes et de rangées du contrôle.
- Le nombre de vendeurs ou de produits suivis peut être aisément étendu en changeant le nombre de lignes et de colonnes de la table et du contrôle. Il n'est pas besoin de modifier l'affichage, du fait des capacités de défilement du contrôle grille.

Une fois que vous aurez créé et exécuté cette application, vous aurez une meilleure notion de l'utilisation du contrôle grille. Vous pourrez aussi voir comment adapter cet exemple simplifié à une utilisation dans d'autres applications.

Créez un nouveau projet et ajoutez les contrôles et les valeurs de propriétés listés au Tableau 22.4.

Pensez à ajouter le contrôle grille à la boîte à outils avant de commencer à travailler sur la feuille. Ajoutez le contrôle Microsoft FlexGrid Control 6.0, comme cela a été expliqué auparavant.

Une fois ces contrôles placés et leurs valeurs paramétrées, vous êtes prêt à saisir le code. N'oubliez pas que les données d'origine résideront dans un tableau dont le nombre de lignes et de colonnes correspond, par choix de conception, à celui du contrôle grille. Vous savez donc déjà, au vu du Tableau 22.4 que la table qui alimentera le contrôle grille aura huit colonnes et vingt lignes.

Tableau 22.4 : Placez ces contrôles et ces valeurs de propriétés pour créer la feuille de commissions de vente

Propriété	*Valeur*
Feuille : `Name`	`frmSales`
Feuille : `Caption`	`Utilisation des tableaux et du contrôle grille`
Feuille : `Height`	`7920`
Feuille : `Width`	`9180`
Label #1 `Name`	`lblSales`
Label #1 `Alignment`	`Center`
Label #1 `Caption`	`Diagramme de contrôle des commissions de vente`
Label #1 `FontSize`	`18`
Label #1 `FontStyle`	`Gras`
Label #1 `Height`	`495`
Label #1 `Left`	`1200`
Label #1 `Top`	`1200`
Label #1 `Width`	`6015`
Label #2 `Name`	`lblInMin`
Label #2 `Alignment`	`Center`
Label #2 `Caption`	`1 %`
Label #2 `FontSize`	`8`
Label #2 `Height`	`255`
Label #2 `Left`	`960`
Label #2 `Top`	`5640`
Label #2 `Width`	`375`
Label #3 `Name`	`lblInMax`
Label #3 `Alignment`	`Center`
Label #3 `Caption`	`15 %`

Tableau 22.4 : Placez ces contrôles et ces valeurs de propriétés pour créer la feuille de commissions de vente *(suite)*

Propriété	*Valeur*
Label #3 `FontSize`	`8`
Label #3 `Height`	`255`
Label #3 `Left`	`2880`
Label #3 `Top`	`5640`
Label #3 `Width`	`375`
Label #4 `Name`	`lblDeMin`
Label #4 `Alignment`	`Center`
Label #4 `Caption`	`1 %`
Label #4 `FontSize`	`8`
Label #4 `Height`	`255`
Label #4 `Left`	`5160`
Label #4 `Top`	`5640`
Label #4 `Width`	`375`
Label #5 `Name`	`lblDeMax`
Label #5 `Alignment`	`Center`
Label #5 `Caption`	`15 %`
Label #5 `FontSize`	`8`
Label #5 `Height`	`255`
Label #5 `Left`	`7080`
Label #5 `Top`	`5640`
Label #5 `Width`	`375`
Contrôle grille : `Name`	`grdSales`
Contrôle grille : `Cols`	`8`
Contrôle grille : `FocusRect`	`flexFocusNone`
Contrôle grille : `Height`	`3015`

Tableau 22.4 : Placez ces contrôles et ces valeurs de propriétés pour créer la feuille de commissions de vente *(suite)*

Propriété	*Valeur*
Contrôle grille : `Left`	`1320`
Contrôle grille : `Rows`	`20`
Contrôle grille : `Top`	`1800`
Contrôle grille : `Width`	`5895`
Barre de défilement horizontale #1 `Name`	`hscIncrease`
Barre de défilement horizontale #1 `Left`	`1080`
Barre de défilement horizontale #1 `Max`	`15`
Barre de défilement horizontale #1 `Min`	`1`
Barre de défilement horizontale #1 `Top`	`5280`
Barre de défilement horizontale #1 `Width`	`2055`
Barre de défilement horizontale #2 `Name`	`hscDecrease`
Barre de défilement horizontale #2 `Left`	`5280`
Barre de défilement horizontale #2 `Max`	`15`
Barre de défilement horizontale #2 `Min`	`1`
Barre de défilement horizontale #2 `Top`	`5280`
Barre de défilement horizontale #2 `Width`	`2055`
Bouton de commande #1 `Name`	`cmdIncrease`
Bouton de commande #1 `Caption`	`&Augmenter de 15 %`
Bouton de commande #1 `Height`	`495`
Bouton de commande #1 `Left`	`1320`
Bouton de commande #1 `Top`	`6000`
Bouton de commande #1 `Width`	`1575`
Bouton de commande #2 `Name`	`cmdDecrease`
Bouton de commande #2 `Caption`	`&Diminuer de 15 %`
Bouton de commande #2 `Height`	`495`

Tableau 22.4 : Placez ces contrôles et ces valeurs de propriétés pour créer la feuille de commissions de vente *(suite)*

Propriété	*Valeur*
Bouton de commande #2 `Left`	`5520`
Bouton de commande #2 `Top`	`6000`
Bouton de commande #2 `Width`	`1575`
Bouton de commande #3 `Name`	`cmdExit`
Bouton de commande #3 `Caption`	`&Quitter`
Bouton de commande #3 `Height`	`495`
Bouton de commande #3 `Left`	`3720`
Bouton de commande #3 `Top`	`6720`
Bouton de commande #3 `Width`	`1215`

Comprendre le code

Le Listing 22.4 contient le code d'initialisation des valeurs générales du programme. C'est une procédure de commande qui, au chargement de la feuille, exécute, par des instructions `Call`, d'autres sous-routines. La majeure partie du code s'occupe de l'initialisation des en-têtes de la grille avec les noms des vendeurs en colonne et les produits en lignes.

Le Listing 22.4 montre un bel exemple de programmation structurée. Si vous avez plus tard des modifications à apporter au programme, vous n'aurez pas à patauger dans plusieurs pages de code d'une procédure. Les procédures appelées ont au contraire chacune un objet. Vous pouvez modifier plus facilement les actions d'une procédure sans interférer avec du code sans rapport.

La programmation structurée est une pratique qui consiste à placer le code correspondant à un objectif précis dans une sous-routine, puis à appeler ce code à partir d'une procédure de commande.

Listing 22.4 : La procédure Form_Load() initialise plusieurs valeurs à l'aide de sous-routines

```
 1:  Private Sub Form_Load()
 2:    ' Définit la justification des cellules de la grille
 3:    ' et assigne les titres des cellules à la ligne fixe et
 4:    ' les en-têtes de colonnes. Initialise en outre la table
 5:    ' des valeurs et l'envoie au contrôle grille.
 6:    '
 7:    Call InitScrolls     ' Initialise les barres de défilement
 8:    Call CenterCells     ' Centre les cellules
 9:    Call SizeCells       ' Spécifie la largeur des cellules
10:    Call Titles           ' Place les titres de colonne et de ligne
11:    Call FillCells        ' Remplit les cellules
12: End Sub
```

Vous pouvez déterminer d'un coup d'œil ce que fait `Form_Load()` en regardant les routines appelées aux lignes 7 à 11. Si le code appelé ne se trouvait pas dans des procédures séparées, mais incorporées dans `Form_Load()`, il serait beaucoup plus difficile de retrouver plus tard le code que vous voulez modifier.

Le Listing 22.5 contient le code des trois premières procédures appelées : `InitScroll()`, `CenterCells()` et `SizeCells()`. Comme toutes les sous-routines générales, ces procédures doivent apparaître avant `Form_Load()`, pour qu'elles résident dans le module (Général) de la fenêtre de code.

Listing 22.5 : Les trois premières procédures appelées par Form_Load() sont utilisées pour configurer la grille

```
 1:  Private Sub InitScrolls()
 2:    ' Configure les deux barres de défilement
 3:    ' à leur valeur maximale. Même si ces valeurs sont
 4:    ' configurées dans la fenêtre Propriétés, cette procé-
 5:    ' dure permet de modifier plus facilement les valeurs
 6:    ' maximales des barres de défilement s'il en est besoin.
 7:    '
 8:    hscIncrease.Value = 15
 9:    hscDecrease.Value = 15
10: End Sub
11:
12: Private Sub CenterCells()
13:    ' Configure la justification des cellules
14:    ' en alignement centré. Assurez-vous de centrer
15:    ' les en-têtes de ligne et de colonne.
16:    '
17:    Dim Column As Integer
18:    '
```

```
19:   ' Commence par centrer les cellules d'en-têtes
20:   For Column = 0 To 7
21:     grdSales.Col = Column   ' configure la colonne courante
22:     ' Centre les cellules fixes de cette colonne
23:     grdSales.ColAlignment(Column) = flexAlignCenterCenter
24:   Next Column
25: End Sub
26:
27: Private Sub SizeCells()
28:   ' Specifie la largeur de chaque cellule
29:   Dim Column As Integer
30:   For Column = 0 To 7
31:     grdSales.ColWidth(Column) = 1100  ' En twips
32:   Next Column
33: End Sub
```

La procédure `InitScrolls()` n'est pas vraiment nécessaire, car vous avez configuré les positions maximales dans la fenêtre Propriétés lorsque vous avez placé les barres de défilement dans la feuille. Cependant, comme cette application change la valeur maximale des barres de défilement à la demande de l'utilisateur, vous pouvez facilement en restaurer les valeurs initiales à partir du code sans avoir à rechercher chaque fois la propriété `Max`.

La procédure `CenterCells()` centre toutes les valeurs des cellules de la grille. La propriété `ColAlignment` exige un numéro de colonne à centrer en indice, fourni en ligne 23. Enfin, la procédure `SizeCells()` à la ligne 27 configure toutes les largeurs de colonnes à 1 100 twips en parcourant la grille et en appliquant la propriété `ColWidth` à chaque colonne (ligne 31).

Le Listing 22.6 montre l'ennuyeux code d'initialisation de la grille. En fait, la majeure partie initialise les titres de la grille, puis remplit une table de valeurs avant de les copier dans la grille. Le tableau à deux dimensions est une zone de stockage intermédiaire qui n'est pas vraiment utile dans cette application particulière. Cependant, en étudiant le code, vous pourrez voir à quel point il est aisé d'afficher les informations d'un tableau multidimensionnel avec un contrôle grille.

Listing 22.6 : Vous devez initialiser les en-têtes de la grille et les cellules de données

```
1:  Private Sub Titles()
2:    ' Remplissage des titres de colonnes
3:    ' Habituellement, ces données proviennent d'une table de base de données
4:    grdSales.Row = 0  ' Tous les noms des vendeurs sont à la ligne 0
5:    grdSales.Col = 1
6:    grdSales.Text = "Smith"
7:    grdSales.Col = 2
8:    grdSales.Text = "Johnson"
```

Listing 22.6 : Vous devez initialiser les en-têtes de la grille et les cellules de données *(suite)*

```
 9:   grdSales.Col = 3
10:   grdSales.Text = "Lake"
11:   grdSales.Col = 4
12:   grdSales.Text = "West"
13:   grdSales.Col = 5
14:   grdSales.Text = "Gates"
15:   grdSales.Col = 6
16:   grdSales.Text = "Kirk"
17:   grdSales.Col = 7
18:   grdSales.Text = "Taylor"
19:   ' Maintenant, remplir les produits
20:   grdSales.Col = 0  ' Tous les noms de produits sont à la colonne 0
21:   grdSales.Row = 1
22:   grdSales.Text = "Gadget #1"
23:   grdSales.Row = 2
24:   grdSales.Text = "Gadget #2"
25:   grdSales.Row = 3
26:   grdSales.Text = "Tube long"
27:   grdSales.Row = 4
28:   grdSales.Text = "Tube court"
29:   grdSales.Row = 5
30:   grdSales.Text = "Règle métallique"
31:   grdSales.Row = 5
32:   grdSales.Text = "Règle en bois"
33:   grdSales.Row = 6
34:   grdSales.Text = "Règle en plastique"
35:   grdSales.Row = 7
36:   grdSales.Text = "Règle en caoutchouc"
37:   grdSales.Row = 8
38:   grdSales.Text = "Panier"
39:   grdSales.Row = 9
40:   grdSales.Text = "Boulon 3C"
41:   grdSales.Row = 10
42:   grdSales.Text = "Boulon 5A"
43:   grdSales.Row = 11
44:   grdSales.Text = "Ecrou 3C"
45:   grdSales.Row = 12
46:   grdSales.Text = "Ecrou 5A"
47:   grdSales.Row = 13
48:   grdSales.Text = "Clou #12"
49:   grdSales.Row = 14
50:   grdSales.Text = "Clou #15"
51:   grdSales.Row = 15
52:   grdSales.Text = "Clou #16"
53:   grdSales.Row = 16
54:   grdSales.Text = "Œillet #4"
55:   grdSales.Row = 17
56:   grdSales.Text = "Œillet #6"
```

```
57:   grdSales.Row = 18
58:   grdSales.Text = "Œillet #8"
59:   grdSales.Row = 19
60:   grdSales.Text = "Joint"
61: End Sub
62:
63: Private Sub FillCells()
64:   ' Remplit les 160 cellules avec des valeurs
65:   ' calculées à partir des valeurs de ligne et de colonne
66:   ' Même si ces données n'ont aucun sens, elles permettent
67:   ' d'insérer rapidement des données dans le tableau et la grille.
68:   '
69:   ' Ces données proviennent normalement d'une base de données.
70:   '
71:   ' Déclarer un tableau de 20 lignes et 7 colonnes qui
72:   ' correspond à la grille sur la feuille. Les indices
73:   ' sont en base zéro, car la grille les utilise aussi.
74:   Dim curData(19, 7) As Currency
75:   Dim Row As Integer
76:   Dim Column As Integer
77:   '
78:   ' Remplir la table de données
79:   For Row = 1 To 19
80:     For Column = 1 To 7
81:       curData(Row, Column) = ((Row + Column) / Row)
82:     Next Column
83:   Next Row
84:   ' Copier le contenu de la table dans la grille
85:   For Row = 1 To 19
86:     For Column = 1 To 7
87:       grdSales.Row = Row
88:       grdSales.Col = Column
89:       grdSales.Text = Format(curData(Row, Column), "###.00")
90:     Next Column
91:   Next Row
92: End Sub
```

La longue liste d'assignations des lignes 4 à 60 ne sert qu'à placer les noms des vendeurs et des produits dans les en-têtes de colonnes et de lignes de la grille. Lorsque vous travaillez sur une grille, il n'y a pas grand-chose d'autre à faire pour l'initialisation des en-têtes. Ces informations proviennent souvent d'une base de données et vous chargerez alors les en-têtes à partir d'une table de la base. Dans le cas de cette application simple, les assignations sont nécessaires.

Les lignes 79 à 83 remplissent la table, dont le nombre de lignes et de colonnes correspond à celui de la grille. Les données sont en fait un calcul réalisé à partir des numéros de lignes et de colonnes, qui ne sert qu'à placer des valeurs différentes dans les éléments du tableau. Ce dernier agit comme une zone de stockage temporaire pour les valeurs de la grille et n'est pas indispensable à cette application. Il permet cependant d'étudier

l'initialisation d'un tableau à partir de boucles imbriquées. Comme des sections précédentes de cette leçon l'ont expliqué, les boucles imbriquées fonctionnent bien pour parcourir les tableaux multidimensionnels. Bien sûr, étant donné qu'une grille fonctionne pratiquement comme une table, les lignes 85 à 91 utilisent des boucles imbriquées pour copier les données dans le contrôle grille.

Remarquez que la ligne 89 utilise la fonction interne `Format()` *pour formater les données afin qu'elles s'affichent dans la grille en francs et en centimes. Vous apprendrez un peu plus loin comment utiliser la propriété de grille* `FormatString()` *pour configurer le format des cellules.*

Le reste du code s'occupe de contrôler les réactions de l'application aux clics de l'utilisateur sur les boutons de commande et les barres de défilement qui apparaissent sous la grille. Ils contrôlent les diverses augmentations et diminutions du prix des commissions. Par exemple, si la commission d'un produit d'un vendeur particulier doit croître ou décroître, l'utilisateur peut sélectionner cette cellule et cliquer sur la barre de défilement pour ne modifier que celle-ci. De plus, les boutons de commande Augmenter de 15 % et Diminuer de 15 % permettent de mettre en place une modification fixe de 15 % de la commission sélectionnée. Le Listing 22.7 montre le code qui permet ces modifications.

Listing 22.7 : Les commissions sont affectées en fonction du contrôle choisi par l'utilisateur

```
 1: Private Sub hscDecrease_Change()
 2:   ' Modifie le titre du bouton de commande
 3:   cmdDecrease.Caption = "&Diminuer de " & Str(hscDecrease.Value) & " %"
 4: End Sub
 5:
 6: Private Sub hscIncrease_Change()
 7:   ' Modifie le titre du bouton de commande
 8:   cmdIncrease.Caption = "&Augmenter de " & Str(hscIncrease.Value) & " %"
 9: End Sub
10:
11: Private Sub cmdIncrease_Click()
12:   ' Augmente les valeurs des cellules sélectionnées
13:   ' en augmentant le pourcentage de la barre de défilement
14:   Dim SelRows As Integer
15:   Dim SelCols As Integer
16:   Dim SelStartRow As Integer
17:   Dim SelStartCol As Integer
18:   Dim RowBeg As Integer
19:   Dim ColBeg As Integer
20:
21:   If (grdSales.HighLight) Then  ' Si sélectionné...
22:     ' Enregistrer les valeurs de cellules sélectionnées
23:     SelStartRow = grdSales.RowSel
```

```
24:     SelStartCol = grdSales.ColSel
25:     RowBeg = grdSales.Row
26:     ColBeg = grdSales.Col
27:     ' Parcourir toutes les cellules sélectionnées
28:     For SelRows = RowBeg To SelStartRow
29:       For SelCols = ColBeg To SelStartCol
30:         grdSales.Row = SelRows
31:         grdSales.Col = SelCols
32:         ' Augmenter la cellule du montant de la barre de défilement
33:       grdSales.Text = grdSales.Text + (hscIncrease.Value /
          ➥100 * grdSales.Text)
34:       grdSales.Text = Format(grdSales.Text, "####.00")
35:      Next SelCols
36:    Next SelRows
37:    ' Restaurer la sélection en surbrillance
38:    grdSales.Row = RowBeg
39:    grdSales.Col = ColBeg
40:    grdSales.RowSel = SelStartRow
41:    grdSales.ColSel = SelStartCol
42:   End If
43: End Sub
44:
45: Private Sub cmdDecrease_Click()
46:   ' Diminue les valeurs des cellules sélectionnées
47:   ' en diminuant le pourcentage de la barre de défilement
48:   Dim SelRows As Integer
49:   Dim SelCols As Integer
50:   Dim SelStartRow As Integer
51:   Dim SelStartCol As Integer
52:   Dim RowBeg As Integer
53:   Dim ColBeg As Integer
54:
55:   If (grdSales.HighLight) Then  ' Si sélectionné...
56:     ' Enregistrer les valeurs de cellules sélectionnées
57:     SelStartRow = grdSales.RowSel
58:     SelStartCol = grdSales.ColSel
59:     RowBeg = grdSales.Row
60:     ColBeg = grdSales.Col
61:     ' Parcourir toutes les cellules sélectionnées
62:     For SelRows = RowBeg To SelStartRow
63:        For SelCols = ColBeg To SelStartCol
64:          grdSales.Row = SelRows
65:          grdSales.Col = SelCols
66:          ' Diminuer la cellule du montant de la barre de défilement
67:          grdSales.Text = grdSales.Text - (hscDecrease.Value /
             ➥100 * grdSales.Text)
68:          grdSales.Text = Format(grdSales.Text, "####.00")
69:        Next SelCols
70:     Next SelRows
71:     ' Restaurer la sélection en surbrillance
72:     grdSales.Row = RowBeg
73:     grdSales.Col = ColBeg
```

Listing 22.7 : Les commissions sont affectées en fonction du contrôle choisi par l'utilisateur *(suite)*

```
74:     grdSales.RowSel = SelStartRow
75:     grdSales.ColSel = SelStartCol
76:   End If
77: End Sub
78:
79: Private Sub cmdExit_Click()
80:   ' Terminer l'application
81:   End
82: End Sub
```

Les fonctions `hscDecrease_Change()` et `hscIncrease_Change()` ne servent qu'à modifier la propriété `Caption` du bouton de commande correspondant quand l'utilisateur clique sur la barre de défilement. Les barres de défilement déterminent de quelle quantité le bouton augmentera ou diminuera les valeurs.

Même si le reste du code, à partir de la ligne 11, est assez long, les deux procédures importantes, `cmdIncrease_Click()` et `cmdDecrease_Click()`, font essentiellement la même chose, si ce n'est que l'une augmente la valeur des cellules sélectionnées, tandis que l'autre la réduit.

Si l'utilisateur ne sélectionne qu'une cellule avant de cliquer sur un bouton de commande, seule cette cellule changera quand la procédure `Click` des boutons de commande s'exécutera.

Etant données les similitudes de ces deux grosses procédures, l'analyse se limitera à `cmdIncrease_Click()`. La ligne 21 s'assure que l'utilisateur a sélectionné au moins une cellule avant d'effectuer l'augmentation de la valeur des cellules. Dans le cas contraire, rien ne se produit. Les lignes 23 à 26 enregistrent la zone rectangulaire de la sélection. Cette plage est définie par le numéro de ligne et de colonne de la cellule en cours (la cellule en haut à gauche de la sélection), et du numéro de ligne et de colonne de la fin de la sélection.

Les lignes 28 et 29 utilisent une boucle `For` imbriquée pour parcourir toutes les cellules de la sélection. Chaque cellule est augmentée de la valeur de la barre de défilement, qui définit le pourcentage de modification des boutons de commande. Les lignes 38 à 41 restaurent la sélection, car si vous changez les valeurs `Row` et `Col` de la grille, toute sélection est effacée ; les valeurs `Row` et `Col` déterminent la cellule active en remplaçant la zone sélectionnée. Les lignes 40 et 41 utilisent `RowSel` et `ColSel` pour remettre la sélection à sa place d'origine.

Exécuter l'application

Maintenant que vous avez créé l'application et ajouté le code, vous pouvez exécuter le programme. Au premier abord, le sens des barres de défilement peut ne pas vous apparaître, ni la manière dont elles sont liées aux boutons de commande mais, après avoir effectué les tâches suivantes, vous apprendrez vite le fonctionnement du programme :

1. Cliquez sur la barre de défilement de gauche pour en diminuer la position (déplacement à gauche). Remarquez que la propriété `Caption` du bouton de commande situé sous la barre change.
2. Cliquez sur une cellule pour la sélectionner dans la grille.
3. Cliquez sur le bouton de commande de droite pour voir la valeur de cette cellule diminuer de 15 %.
4. Sélectionnez une plage de cellules (voir Figure 22.10)
5. Cliquez sur le bouton de commande de gauche pour modifier les cellules sélectionnées de la valeur affichée sur le bouton de commande. La sélection reste affichée pour le cas où vous voudriez continuer à augmenter les commissions de la plage de cellules.
6. Sélectionnez une autre plage et diminuez-en les valeurs.
7. Cliquez sur Quitter pour terminer le programme..

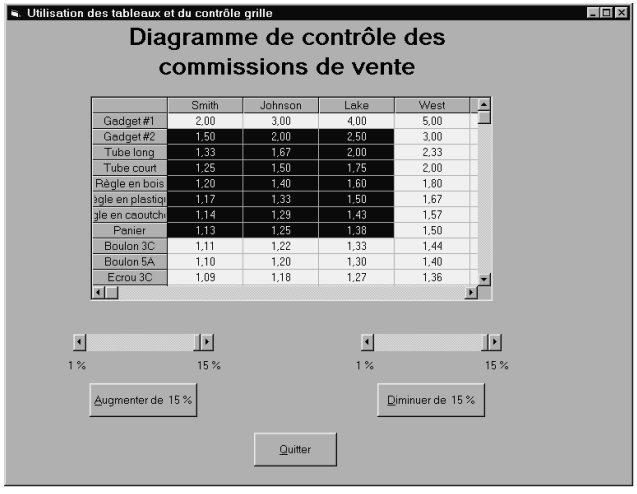


Figure 22.10
Quel que soit le nombre de cellules sélectionnées, vous pouvez augmenter ou diminuer les valeurs de la sélection en cliquant sur un bouton.

La propriété *FormatString*

Au lieu d'utiliser la fonction interne `Format()` pour formater les données lorsque vous les placez sur une grille, vous pouvez paramétrer le format d'une cellule ou d'une plage de manière que toutes les données placées ultérieurement soient formatées de manière conforme. La propriété `FormatString` du contrôle grille demande une donnée chaîne définissant l'apparence du contenu de la cellule. Si `FormatString` est assez obscure au début, une fois maîtrisée, elle permet de formater les données de grille plus vite.

Vous pouvez entrer une valeur `FormatString` dans la boîte de dialogue Pages de propriétés qui s'affiche quand vous cliquez sur la propriété (Personnalisé) du contrôle grille. Cliquez sur l'onglet Style et entrez la valeur dans la zone de texte Format. Vous pouvez aussi assigner la propriété `FormatString` directement, comme le montre cette section.

Une des meilleures caractéristiques de `FormatString` est qu'elle permet de paramétrer les en-têtes de lignes et de colonnes, ce qui évite d'avoir à assigner des valeurs comme dans l'application de la section précédente.

Les règles suivantes déterminent le contenu de la propriété `FormatString` :

- Le symbole séparateur (|) sépare les différents segments de `FormatString`.
- Chaque segment définit une colonne.
- Le texte de chaque segment se comporte comme la valeur d'en-tête de la ligne 0.
- Le texte de chaque segment définit la largeur de cette colonne sur la grille.
- Le caractère < justifie à gauche le texte du segment, ^ le centre, et > le cadre à droite.
- Le texte qui suit un point-virgule (;) définit les en-têtes de colonne 0, affichés à gauche de la ligne.
- Le texte de l'en-tête de ligne le plus long détermine la largeur des titres de la colonne 0.
- Le nombre de segments détermine le nombre maximum de colonnes de la grille.

Si la propriété `FormatString` définit les lignes et les colonnes, comme les en-têtes et leur largeur, vous devez toujours dimensionner le contrôle grille dans la fenêtre feuille pour que l'utilisateur voie toutes les lignes et les colonnes. `FormatString` ne peut en aucun cas redimensionner la grille.

L'exemple suivant génère la grille illustrée à la Figure 22.11 :

```
Dim strCol As String
Dim strRow As String
' Définit les en-têtes de colonne de la grille
```

```
' Le symbole ^ entraîne le centrage de tous les en-têtes
' qui suivent. Le symbole de séparation définit chaque
' colonne.
strCol = "^|Smith|Johnson|Lake|West|Gates|Kirk|Taylor"
' Définit les en-têtes de ligne de la grille en créant une
' chaîne. Le point-virgule indique à VB de définir ces
' valeurs en tant qu'en-têtes de ligne.
strRow = ";|Gadget #1|Gadget #2|Tube long|Tube court|"
strRow = strRow & "Règle métallique|Règle en bois|Règle en
➥plastique|"
strRow = strRow & "Règle en caoutchouc|Panier|Boulon 3C|Boulon
➥5A|"
strRow = strRow & "Ecrou 3C|Ecrou 5A|Clou #12|Clou #15|"
strRow = strRow & "Clou #16|Œillet #4|Œillet #6|"
strRow = strRow & "Œillet #8|Joint"
' Formate la grille en assignant les deux chaînes
grdSales.FormatString = strCol & strRow
```

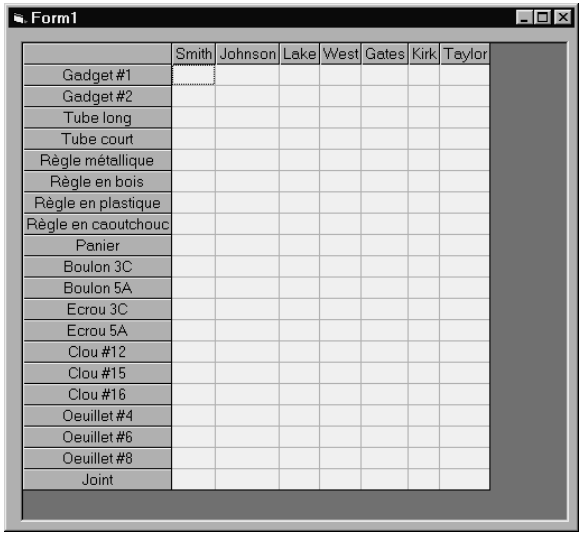


Figure 22.11
Utilisez la propriété FormatString pour formater les en-têtes de ligne et de colonne.

La grille est identique à celle qui a été créée dans l'application précédente, à la différence que les largeurs des colonnes de la grille de la Figure 22.11 sont les largeurs de leur en-tête respectif.

`FormatString` permet à l'évidence d'éliminer de nombreuses lignes de code lourd dans l'application précédente. Le Listing 22.6 contient de nombreuses instructions d'assignation des en-têtes de ligne et de colonne. La propriété `FormatString` peut prendre en charge les en-têtes bien plus efficacement.

Enregistrer des images dans le contrôle grille

Le contrôle grille permet d'enregistrer des images bitmap et des icônes dans les cellules en utilisant la propriété `CellPicture`. Pour placer une image dans une cellule quelconque, suivez ces étapes :

1. Assignez les propriétés `Row` et `Col` à l'indice de la cellule dans laquelle vous voulez voir s'afficher l'image.
2. Assignez les propriétés `ColWidth` et `RowHeight` à la largeur et à la hauteur de l'image pour contrôler la taille de l'image.
3. Assignez l'image à la propriété `CellPicture`.

Vous ne pouvez pas assigner une image lors de la conception.

Si vous placez d'abord votre image dans un contrôle image de la feuille en paramétrant sa propriété `Visible` à `False`, vous pouvez assigner le contrôle image à la cellule au lieu d'avoir à utiliser `LoadPicture()` en spécifiant le nom de fichier exact de l'image à placer dans la cellule. Le contrôle image permet de paramétrer plus facilement la largeur et la hauteur de la cellule pour qu'elles correspondent à l'image.

Supposons que vous voulez placer une image dans la cellule en haut à gauche du contrôle grille. Cette cellule ne contient, en général, pas de texte d'en-tête. Le code suivant permet de le faire :

```
' Rend la cellule en haut à gauche courante
grdSales.Row = 0
grdSales.Col = 0
' Paramètre la hauteur et la largeur de la cellule
' pour qu'elle corresponde à l'image
grdSales.ColWidth(0) = Image1.Width
grdSales.RowHeight(0) = Image1.Height
' Assigne l'image à la cellule
Set grdSales.CellPicture = Image1.Picture
```

Remarquez qu'il faut utiliser une instruction `Set` pour assigner l'image à la cellule. Une instruction d'assignation normale ne suffit pas, car seul le chemin d'accès à l'image serait assigné dans ce cas à la cellule. La Figure 22.12 montre à quoi cela peut ressembler.

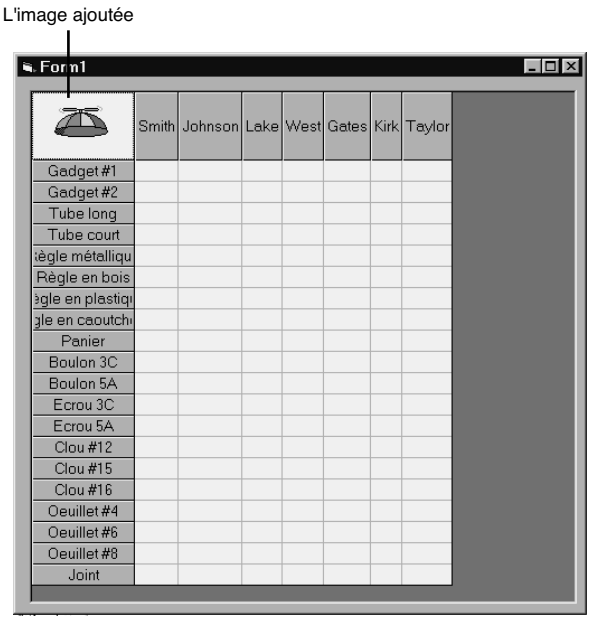


Figure 22.12
Les images peuvent enjoliver un contrôle grille.

En résumé

Cette leçon supplémentaire vous a expliqué comment configurer des tableaux multidimensionnels pour vous permettre de gérer plus facilement les données. Le tableau multidimensionnel le plus courant, la table à deux dimensions, apparaît dans de nombreux types de situations où vous devez suivre des lignes et des colonnes de données, lorsque vous avez, par exemple, à gérer un système d'inventaire des prix. Visual Basic permet de suivre jusqu'à soixante dimensions de données, mais vous aurez rarement à gérer plus de trois ou quatre dimensions.

Une manière simple de représenter les données de tableaux est le contrôle grille. Il en existe plusieurs types que vous trouverez dans la boîte de dialogue Composants qui s'affiche quand vous sélectionnez l'option de menu Projet, Composant. Le contrôle FlexGrid en est un exemple des plus classiques. Cette leçon vous a expliqué comment le configurer et le gérer.

Questions-réponses

Q Comment puis-je savoir ce que représentent les indices quand je déclare un tableau à quatre ou cinq dimensions ?

R Chaque indice peut représenter ce que vous souhaitez lui faire représenter. Par exemple, dans un tableau à deux dimensions, le premier indice n'a pas à indiquer des lignes, ni le second des colonnes ; vous pouvez aussi bien les inverser. Ce n'est que l'approche la plus courante de dire que l'indice de droite représente les colonnes, l'indice à sa gauche des lignes, et l'indice à leur gauche des couches de tables (comme dans un cube). Mais il est conseillé de suivre ce standard. Si une autre personne doit un jour faire de la maintenance sur votre code, vous lui simplifiez le travail en suivant les conventions d'utilisation des valeurs de tableaux.

Même si les gens ne visualisent pas clairement plus de trois dimensions, un quatrième indice peut permettre de garder une trace du nombre de tableaux à trois dimensions réservés par les trois indices de droite. Un cinquième indice pourrait alors représenter le nombre de groupes de tableaux à trois dimensions, etc. Si vous augmentez le nombre de dimensions au-delà de trois, votre programmation n'en devient pas pour autant beaucoup plus difficile, mais se représenter chaque indice peut l'être.

Q Aurai-je souvent à utiliser un tableau de plus de trois ou quatre dimensions ?

R Rarement et peut-être même jamais.

Q Si je n'utilise probablement jamais des tableaux multidimensionnels de plus de trois ou quatre dimensions, pourquoi les étudier ?

R En fait, quand vous comprendrez les tableaux à plusieurs dimensions (et que vous découvrirez que l'ajout d'une nouvelle dimension, en termes de stockage machine, ne fait que vous donner des occurrences supplémentaires du tableau que les dimensions précédentes déclarent), vous n'aurez pas ajouté tant de complexité que cela. Le fait que vous n'utilisiez que peu des tableaux de plus de quatre dimensions ne signifie pas qu'il ne faut pas apprendre combien il est simple d'ajouter une dimension. Cependant, n'oubliez pas que chaque dimension multiplie d'autant la quantité de mémoire que vous utilisez. Ne déclarez pas un tableau trop important sans prendre en compte la mémoire disponible sur la machine de destination. Certaines applications de modélisation scientifique et mathématique exigent plusieurs dimensions. Si vous programmez dans des domaines techniques, vous risquez d'avoir à utiliser cinq dimensions, ce qui sera moins probable si vous n'écrivez que des applications de gestion.

Q Puis-je utiliser `FormatString` pour ajouter des données au contrôle grille ?

R Non. `FormatString` ne sert qu'à ajouter des valeurs d'en-tête de ligne et de colonne à la grille. Vous devez assigner les valeurs de données qui s'affichent à l'intérieur de la grille.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités, ainsi que des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se situent à l'Annexe A.

Quiz

1. Vrai ou faux. Tous les éléments d'une ligne d'un tableau multidimensionnel doivent être du même type, mais les différentes lignes peuvent être de type différent.
2. Etant donné la déclaration suivante d'un tableau multidimensionnel, quel indice — le premier, le second ou le troisième — spécifie habituellement les lignes ?

```
Dim sngArray(8, 9, 10)
```

3. Etant donné le tableau d'entiers suivant (nommé `intAra`), quelles valeurs représentent les éléments suivants, en considérant qu'une instruction `Option Base 1` est présente dans le programme ?

```
a. intAra(1, 1)

b. intAra(3, 2)

c. intAra(2, 3)
```

```
 4   1   3   5   9
10   2  12   1   6
25  43   2  91   8
14   7  28  71  14
```

4. Vrai ou faux. Visual Basic supporte jusqu'à soixante dimensions.
5. Vrai ou faux. Vous pouvez utiliser la fonction `Array()` pour initialiser un tableau multidimensionnel à l'aide d'une instruction.
6. Quel type de contrôle est bien adapté à l'affichage de données de tables ?

7. A quoi servent les lignes et les colonnes fixes d'un contrôle grille ?
8. Comment pouvez-vous assigner les cellules d'une table aux cellules d'un contrôle grille ?
9. Vrai ou faux. Pour assigner une image à une cellule, vous pouvez utiliser la propriété `CellPicture` à la conception, comme vous le faites pour d'autres contrôles.
10. Qu'est-ce qui est le plus efficace ? Utiliser `FormatString` ou des instructions d'assignation pour configurer les en-têtes de la grille.

Exercices

1. Calculez le nombre d'éléments que réservent les instructions suivantes :

```
Option Base 1
Dim intAra(4, 7) As Integer
```

2. Si vous omettez l'instruction `Option Base` dans le module de déclaration de la procédure, calculez le nombre d'éléments réservés par l'instruction suivante :

```
Dim intAra(4, 7) As Integer
```

3. Modifiez l'application de grille des vendeurs de cette leçon pour que les valeurs de la grille soient récrites dans la table avant que le programme se termine. Même si on ne fait rien de cette table, cet exercice est une bonne manière de montrer la relation univoque entre la table et la grille. Il fournit aussi un endroit où vous pourriez écrire ces valeurs dans une base de données si vous aviez besoin de sauvegarder les modifications de commissions faites par l'utilisateur.

Chapitre 23

L'API Windows

Cette leçon supplémentaire décrit comment accéder aux routines internes de Windows. Si Visual Basic est capable de faire à peu près tout ce que vous voulez, certaines applications exigent des caractéristiques qui ne peuvent être obtenues qu'au prix d'une programmation lourde. Cependant, vous pouvez utiliser dans votre application Visual Basic des routines déjà disponibles ailleurs dans Windows — comme celles que vous pouvez écrire en C ou C++ et enregistrer dans des DLL.

En exploitant ces routines de Windows, vous pouvez étendre la puissance de Visual Basic et lui faire effectuer certaines fonctions que seul Windows a réellement autorité à exécuter. Cette leçon décrit de nombreuses routines utiles et la manière d'y accéder. Si votre application doit gérer un curseur de Windows, par exemple, il existe déjà des routines internes pour cela ; il peut donc être plus simple d'y faire appel depuis votre application Visual Basic.

Vous apprendrez aujourd'hui :

- ce qu'est l'API Windows ;
- pourquoi votre application peut avoir besoin de routines Windows absentes de Visual Basic ;
- les DLL (Dynamic Link Library) ;
- comment connecter Visual Basic aux routines de l'API Windows avec l'instruction `Declare` ;

- plusieurs nouvelles procédures utilisables avec les routines de l'API Windows ;
- comment éviter les problèmes en spécifiant des routines de l'API ;
- comment interpréter et utiliser les nouveaux types d'arguments qu'exigent les routines de l'API Windows ;
- comment créer des fonctions d'appel des routines API les plus courantes pour que votre application Visual Basic y accède plus facilement.

L'API Windows

L'*API Windows* est un ensemble de routines à la disposition du programmeur Visual Basic. D'une certaine manière, ces routines de l'API fonctionnent comme les fonctions internes de Visual Basic. Quand vous avez besoin d'utiliser le code d'une routine de l'API, votre programme Visual Basic l'appelle. Quand l'API Windows se termine, le contrôle revient à votre programme pour lui permettre de continuer.

Pratiquement, tout ce que vous pouvez faire dans Windows est réalisable dans un programme Visual Basic en appelant la routine de l'API Windows adaptée. Vous pouvez même déclencher une réinitialisation de la machine.

L'API Windows (Application Programming Interface, ou interface de programmation d'application, est un ensemble de routines de Windows que vous pouvez appeler depuis Visual Basic.

Toutes les routines de l'API Windows sont enregistrées dans des fichiers particuliers nommés DLL. Plusieurs milliers de routines de l'API sont disponibles. Elles se situent dans des fichiers enregistrés dans les dossiers Windows et Windows\System. Les fichiers DLL sont installés avec Windows ; vous avez donc automatiquement accès à ces bibliothèques.

Une DLL, (Dynamic Link Library), ou bibliothèque de liens dynamiques, est un ensemble de routines de l'API disponibles aux applications écrites en Visual Basic ou en un autre langage qui supporte l'utilisation des DLL.

La plupart des DLL ont une extension de nom de fichier .DLL ou .EXE. Tout programme que vous écrivez a accès aux DLL de Windows. Elles faisaient déjà partie des versions antérieures de Windows (avant Windows 95), mais leur nom ne comportait pas "32", ce qui signale qu'elles sont compatibles 32 bits. Les versions antérieures à Windows 95 étaient compatibles 16 bits, ce qui signifie que les données transitaient dans le système 16 bits (soit deux octets) à la fois. La programmation sous un environnement 32 bits procure une souplesse, une vitesse et une efficacité supérieures.

Voici les trois DLL les plus courantes :

- **USER32.DLL.** Elle contient des fonctions qui contrôlent l'environnement Windows et l'interface utilisateur, par exemple les curseurs, les menus et les fenêtres.
- **GDI32.DLL.** Elle contient des fonctions qui contrôlent les sorties à l'écran et sur d'autres périphériques.
- **KERNEL32.DLL.** Elle contient des fonctions qui commandent l'interface matériel et logiciel interne de Windows. La plupart des routines de service de mémoire, de fichier et de répertoire se situent dans KERNEL32.DLL.

Kernel signifie Noyau, et GDI (Graphics Device Interface), interface de périphérique graphique.

Windows est un système d'exploitation en plusieurs couches, qui commence avec celle que voit l'utilisateur (l'interface utilisateur graphique ou GUI), et qui se termine avec la couche la plus proche du matériel, qui contrôle le flux des données entre les programmes et le matériel. Cette couche basse du système d'exploitation est nommée le noyau (kernel). D'où le nom KERNEL32.DLL donné à la bibliothèque de liens dynamiques contenant les routines du noyau.

Ces trois fichiers contiennent la majorité des routines, ou fonctions, de l'API que vous pouvez appeler depuis vos applications Visual Basic. Si vous parcourez les dossiers Windows et Windows\System, vous verrez d'autres DLL, par exemple COMDLG.DLL, MAPI32.DLL, NETAPI32.DLL et WINMM.DLL. Chaque fois que Microsoft ajoute des fonctionnalités au système d'exploitation, de nouvelles DLL apparaissent.

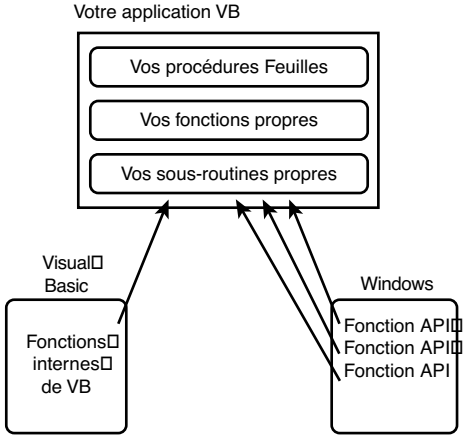
Les DLL ne sont pas uniquement des parties de Windows. Lorsque vous ajoutez de nouvelles applications au système, elles fournissent souvent leurs propres DLL. Au fur et à mesure des installations, de nombreux fichiers DLL apparaissent dans la machine.

Les DLL vous donnent un pouvoir sur votre système bien plus important que celui que propose normalement Visual Basic. Quand vous utilisez une fonction de l'API Windows, vous travaillez au cœur du système d'exploitation. Comme toujours, un pouvoir entraîne une responsabilité. L'environnement et le débogueur Visual Basic reconnaissent les fonctions internes normales de Visual Basic, mais les fonctions de l'API sont bien au-delà de leur portée. Vous risquez donc un crash du système et la perte de tout votre travail, simplement en exécutant une application Visual Basic qui spécifie un argument incorrect dans un appel de fonction de l'API Windows.

Enregistrez souvent votre projet si vous faites des appels aux fonctions de l'API. Ainsi, si vous appelez par erreur une fonction de l'API qui entraîne un crash du système, vous ne perdrez pas tout ce que vous avez fait.

La Figure 23.1 montre comment une routine API apparaît dans votre programme Visual Basic. Les routines API proviennent du système d'exploitation et sont distinctes de Visual Basic.

Figure 23.1
Les routines API demeurent dans le système d'exploitation.



Nature des DLL

Le terme *lien dynamique* a une signification particulière pour les programmeurs. Une routine *liée dynamiquement* à un programme, que ce soit une sous-routine ou une fonction, n'est connectée au programme qu'après la compilation de ce dernier. La fonction n'est disponible qu'au moment de l'exécution. Les fonctions que vous écrivez dans la fenêtre de code sont dites *liées statiquement*, ce qui signifie qu'elles sont combinées avec le reste de votre code source lors de la compilation du programme. Les fichiers DLL ne se fusionnent pas à vos programmes. Le programme y a accès lors de l'exécution, mais son fichier EXE ne contient jamais physiquement les routines de DLL.

La différence est cruciale quand il s'agit d'utiliser des fonctions situées dans des bibliothèques de liens dynamiques, car ni la bibliothèque, ni les fonctions que votre application appelle ne sont considérées comme faisant partie de votre programme. Les fonctions API ne viennent jamais augmenter la taille du fichier de votre application. Lors de l'exécution du programme, ces routines ne sont chargées que le temps de leur exécution ; puis, si elles ne sont plus nécessaires, le système d'exploitation Windows

peut libérer leurs ressources pour donner plus de mémoire et de temps processeur aux nouvelles routines qui peuvent démarrer.

Le grand avantage de ces liens dynamiques n'est cependant pas une utilisation efficace des ressources. Si Windows est modifié, de nouvelles DLL remplacent les anciennes. Vos applications sont donc à même de supporter les nouvelles caractéristiques sans qu'il soit besoin de recompiler chaque application qui utilise l'API Windows. Par exemple, si vous vous en souvenez, Windows 95 a modifié la présentation des fenêtres. Les icônes dans le coin supérieur droit de la fenêtre sont différentes de ce qu'elles étaient dans Windows 3.11. Un programme Visual Basic qui appelle l'API Windows pour afficher une fenêtre fonctionne dans chaque environnement Windows. Sous Windows 3.11, il affiche les anciennes icônes et sous Windows 95, les nouvelles, alors que le programme n'est pas modifié. Donc, dans la plupart des cas, un programme qui accède à l'API Windows ne demande pas de modifications lorsque vous changez de version de Windows.

Windows n'est pas composé d'un seul gros programme. C'est en fait une collection de nombreux programmes, dont certains résident dans des fichiers DLL. Windows est sans nul doute le plus gros utilisateur des DLL.

L'utilisation des routines en DLL présente l'avantage de permettre à plusieurs programmes qui s'exécutent sous Windows d'accéder aux mêmes routines des fichiers DLL. De plus, tous les utilisateurs disposent des routines de DLL standard : Windows étant nécessaire pour pouvoir exécuter une application Visual Basic, les DLL nécessaires seront donc disponibles.

L'instruction *Declare*

L'appel des routines de l'API Windows nécessite une instruction spéciale : `Declare`. Les fonctions internes de Visual Basic n'ont pas besoin d'instruction `Declare`, car il sait comment fonctionnent ses propres fonctions et il connaît les arguments nécessaires à chacune. Mais, les routines API étant en dehors de la portée de Visual Basic, vous devez utiliser `Declare` pour l'informer sur la fonction de l'API que vous appelez.

L'instruction `Declare` effectue les tâches suivantes :

- Spécifie où se situe la fonction de l'API.
- Identifie les arguments nécessaires à la fonction de l'API en nombre et type de données.
- Indique si la fonction de l'API renvoie une valeur.

L'emplacement de l'instruction `Declare` a une incidence sur la manière dont votre application Visual Basic gère la fonction ; elle détermine de quelles parties de l'application on peut appeler la fonction décrite :

- Si vous déclarez la routine de l'API Windows dans le module de feuille, hors de sa section de déclaration générale (par exemple, dans une procédure événementielle), seul le code du module peut appeler la routine API. L'instruction `Declare` doit désigner la routine comme privée à l'aide du mot clé `Private`.
- Si vous déclarez la routine de l'API Windows dans la section de déclarations générales d'un module ou d'une feuille, elle sera à la disposition de toute l'application. On dit qu'elle a une *portée publique* sur tous les modules de l'application. Utilisez le mot clé `Public` pour l'indiquer.

Comme pour toute procédure, une routine de l'API Windows peut être une fonction ou une sous-routine, selon qu'elle renvoie une valeur ou pas. Le format suivant décrit la version procédure de l'instruction `Declare` :

```
Declare Sub procName Lib "libName" [Alias "alias"] [([[ByVal]
➥var1 [As dataType][,[ByVal] var2 [As dataType]] ... [,[ByVal]
➥varN [As dataType])]
```

L'instruction `Declare` indique à Visual Basic le type de procédure de l'API (sous-routine ou fonction), le nom de la routine, le nom de fichier de la bibliothèque DLL dans laquelle la routine est enregistrée (par exemple, KERNEL32.DLL), les arguments et leurs types de données. Si la routine est une fonction, `Declare` décrit aussi le type de données renvoyé.

Comme pour la plupart des instructions, le format de `Declare` donne des appréhensions, mais son utilisation réelle est légèrement plus simple que son format le laisse paraître. Cependant, vous devez toujours être très attentif pour faire correspondre tous les arguments et les valeurs requis à ceux de la routine API que vous appelez afin qu'elle s'exécute correctement.

Le format suivant décrit la version fonction de l'instruction `Declare`. Il ne diffère de la forme sous-routine que par le mot clé `Function` et le type de donnée renvoyé à la fin de l'instruction :

```
Declare Function procName Lib "libName" [Alias "alias"]
➥[([[ByVal] var1 [As dataType][,[ByVal] var2 [As dataType]] ...
➥[,[ByVal] varN [As dataType])] As dataType
```

Voici quelques exemples qui illustrent l'instruction `Declare` (vous trouverez ces instructions dans le module général Module1 du projet exemple CallDlls.VBP fourni avec Visual Basic) :

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias
```

```
➥"GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As
➥Long) As Long
Declare Sub GetSystemInfo Lib "kernel32" (lpSystemInfo As
➥SystemInfo)
```

Vous remarquerez que certaines déclarations de DLL sont assez longues. Les fonctions internes demandent différents nombres d'arguments, c'est aussi le cas des déclarations et des appels de DLL.

Vous devez absolument respecter le nom exact de la routine de l'API Windows, avec les majuscules et les minuscules. Ces fonctions de l'API sont en fait des routines en langage C et Visual Basic doit utiliser une syntaxe reconnue par le C pour que les fonctions opèrent correctement. Si vous ne respectez pas la casse des lettres ou si vous utilisez un format différent, l'appel échouera.

Comprendre les types de données de l'API

Une des raisons de la difficulté des appels des routines API est que Windows utilise un ensemble de types de données légèrement différent de Visual Basic. Si l'API Windows utilise les types de données Long et String, elle en utilise aussi d'autres comme RECT et MSG. Trouver le format exact peut parfois être difficile.

Vos arguments doivent non seulement correspondre à la liste des arguments nécessaires de l'API en nombre et type de données, mais vous devez aussi les transmettre de la bonne manière — soit en valeur, soit par référence. Utilisez le mot clé `ByVal` lorsque c'est nécessaire, sinon `ByRef` est pris par défaut. Dans une même routine API, certains arguments demandent des méthodes différentes.

Le mot clé `Alias` est utilisé dans une instruction `Declare` pour convertir certaines chaînes comportant des caractères illégaux ou des noms de routines API non autorisés, en un équivalent accepté par Visual Basic, par exemple `_lopen` (nom d'API correct, mais qui est invalide en tant que nom de procédure Visual Basic).

Vous rencontrerez des types de données étranges que vous pouvez ne pas reconnaître. Le Tableau 23.1 en décrit certains et liste les données qui diffèrent des types de Visual Basic.

Le Tableau 23.1 ne contient que quelques types de données que vous trouverez dans les instructions `Declare` de l'API Windows. Etant donné ces routines API particulières et leurs nombreux arguments, comment est-il possible de savoir laquelle utiliser ? La section suivante vous explique comment utiliser un outil fourni avec Visual Basic pour gérer les routines API.

Tableau 23.1 : Types de données particuliers utilisés par les routines API

Type de données	*Description*
ANY	Une routine de l'API Windows qui accepte différents types de données les indiquera par ANY. Tous les arguments de type ANY étant transmis par référence, il ne faut pas utiliser le mot clé `ByVal`.
ATOM	Entier. Toujours transmis par valeur et décrit dans une déclaration de routine API sous la forme `ByVal` *`argument`*`%` ou `ByVal` *`argument`* `As Integer`.
BOOL	Entier long. Toujours transmis par valeur et décrit dans une déclaration de routine API sous la forme `ByVal argument%` ou `ByVal` *`argument`* `As Long`.
CHAR	Byte. Toujours transmis par valeur et décrit dans une déclaration de routine API sous la forme `ByVal` *`argument`* `As Byte`.
COLOREF	Entier long utilisé pour spécifier des valeurs de couleur. Tou7jours transmis par valeur et décrit dans une déclaration de routine API sous la forme `ByVal` *`argument`*`%` ou `ByVal` *`argument`* `As Long`.
DWORD	Entier long. Toujours transmis par valeur et décrit dans une déclaration de routine API sous la forme `ByVal` *`argument`*`%` ou `ByVal` *`argument`* `As Long`.
NULL	Entier long utilisé pour les valeurs non initialisées. Décrit dans une déclaration de routine API sous la forme `ByVal` *`argument`*`%` ou `ByVal` *`argument`* `As Long`.
LPSTR, LPCSTR	Correspond au type de données String. Décrit dans une déclaration de routine API sous la forme `ByVal` *`argument`*`$` ou `ByVal` *`argument`* `As String`.
STRUCTURE	Vous tomberez parfois sur d'étranges types de données API comme RECT, MSG et UDT. Ils définissent des types de données complexes, les structures, qui sont un regroupement de plusieurs autres types de données. Chaque routine API qui utilise une structure demande une mise en forme particulière et il faut étudier les arguments obligatoires de la routine pour savoir comment la formater.

Si une routine API exige un type de données String, vous devez transmettre une chaîne définie de longueur fixe avec beaucoup de bourrage. Par exemple, doublez la longueur de la chaîne la plus longue que vous vous attendez à voir renvoyée par la routine API, puis déclarez un argument chaîne de longueur fixe avec autant d'espaces avant de le transmettre à la routine API. Ne vous souciez pas de la longueur de la chaîne si elle n'est pas modifiée par la routine API.

La Visionneuse d'API

Windows contient des milliers de routines API qui peuvent être appelées. Connaître leur format, même d'un petit nombre, est difficile. C'est pourquoi Visual Basic contient un outil particulier, la *Visionneuse d'API*, qui peut être utilisé pour obtenir de l'aide sur le format des routines API.

La Visionneuse d'API affiche les procédures de l'API et les groupe par sujet pour que vous puissiez trouver la routine dont vous avez besoin.

La Visionneuse d'API permet de retrouver les routines et les arguments API, puis de copier et coller ces informations dans la fenêtre de code. Suivant votre installation de Visual Basic, la Visionneuse d'API peut se lancer de deux manières.

Le bouton Copier de la Visionneuse d'API copie les informations de déclaration sélectionnées dans le Presse-papiers de Windows. De plus, si vous cliquez sur l'option `Public` *ou* `Private` *avant de cliquer sur Copier, il inclut le mot clé de qualification correspondant dans l'instruction* `Declare`*, ce qui évite une modification manuelle.*

Dans certains cas, la Visionneuse d'API est installée dans le menu Démarrage. Pour le vérifier, sélectionnez Démarrage, Programmes, Microsoft Visual Basic 6.0, Outils Microsoft Visual Studio 6.0, Visionneuse d'API. Si elle ne s'y trouve pas, vous pourrez peut-être la lancer à partir de l'environnement Visual Basic. Pour cela, sélectionnez Compléments, Gestionnaire de compléments pour afficher la boîte de dialogue correspondante. Double-cliquez sur la rubrique Visionneuse d'API, si elle existe, pour l'ajouter à votre menu Compléments. Vous pouvez démarrer le programme en sélectionnant Compléments, Visionneuse d'API.

La Figure 23.2 montre la fenêtre de la Visionneuse d'API qui s'affiche.

Il se peut que vous ne puissiez toujours pas démarrer la Visionneuse d'API. Dans ce cas, elle n'est peut-être pas installée sur votre machine. Vous devrez ouvrir le Panneau de configuration et sélectionner l'icône Ajout/Suppression de programmes. Sélectionnez l'entrée Microsoft Visual Basic et installez la Visionneuse d'API à partir de l'entrée Outils de la liste. Il vous faudra le CD-ROM d'installation de Visual Basic pour achever l'installation.

La Visionneuse d'API extrait les informations sous-jacentes des fichiers texte Mapi32.txt et Win32api.txt qui sont installés avec lui.

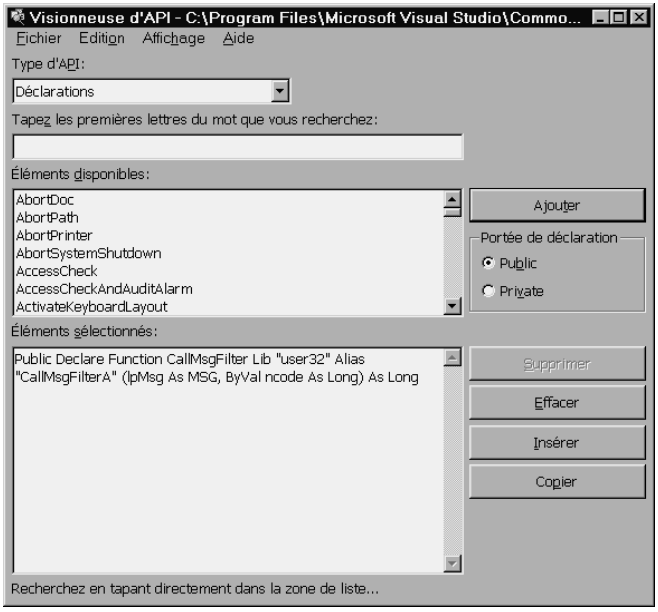


Figure 23.2
La Visionneuse d'API permet de déterminer plus facilement le format des routines API.

Comme la plupart des routines API qui vous intéressent sont situées dans le fichier Win32api.txt, sélectionnez Fichier, Charger le fichier texte et sélectionnez ce fichier. La Visionneuse d'API peut convertir le fichier texte en une base de données Access (avec l'extension .MDB) si vous sélectionnez l'option de menu Convertir le texte en base de données (voir Figure 23.3). Une fois la conversion faite, le fichier pourra être chargé ultérieurement à partir de l'option de menu Fichier, Charger le fichier base de données.

La liste déroulante du haut est nommée Type d'API. Si vous l'ouvrez, vous y trouverez ces trois valeurs :

- **Constantes**. Liste toutes les constantes nommées reconnues par le fichier API Windows chargé.
- **Déclarations**. Liste toutes les déclarations qui apparaissent dans le fichier API chargé.
- **Types**. Liste tous les types de données reconnus par le fichier API chargé.

La zone de liste Eléments disponibles contient toutes les routines de l'API Windows du fichier chargé et les types de valeur. Par exemple, afin de trouver l'instruction `Declare` nécessaire pour la routine API `GetWindowsDirectory`, suivez ces étapes :

1. Sélectionnez Déclarations dans la zone de liste Type d'API. Une quantité de candidats s'affichent dans la liste Eléments disponibles.

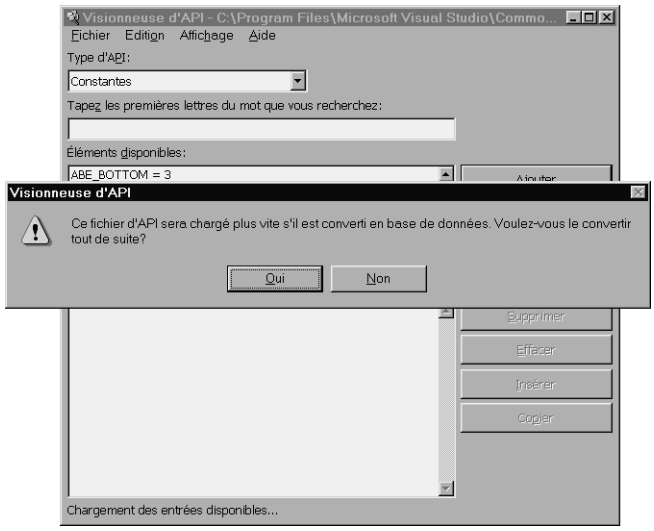


Figure 23.3
La Visionneuse d'API peut enregistrer les informations sous-jacentes dans une base de données pour un accès plus rapide.

2. Vous pouvez trouver rapidement une déclaration particulière en en tapant les premières lettres dans la zone de texte. Tapez `getw`, et tous les éléments commençant par ces lettres s'affichent.
3. Faites défiler jusqu'à la rubrique `GetWindowsDirectory`.
4. Double-cliquez sur l'entrée pour afficher l'instruction `Declare` correspondant à cette fonction (voir Figure 23.4).

Vous pouvez maintenant sélectionner et copier l'instruction `Declare` complète, puis la coller dans votre code.

Appel d'une routine API simple

Avant d'en apprendre plus sur l'API Windows, vous pouvez souhaitez voir l'une de ces routines API en action. Une des procédures API les plus simples est la fonction `MessageBeep`. Elle réalise ces deux actions :

- Si l'argument transmis à la fonction est positif, un bip est émis sur la *carte son* du PC.
- Si l'argument transmis à la fonction est négatif, un bip est émis sur le *haut-parleur* du PC.

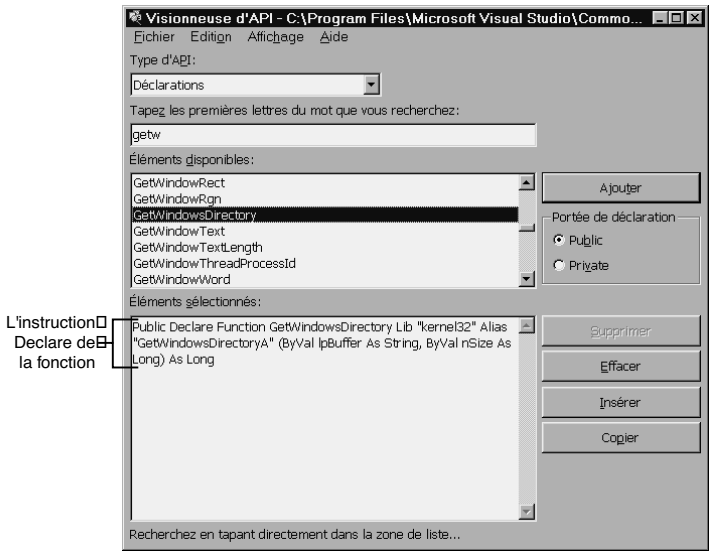


Figure 23.4
La Visionneuse d'API affiche l'instruction Declare requise par l'instruction sélectionnée.

Vous pouvez bien sûr utiliser plus simplement la commande Visual Basic `Beep` pour déclencher un bip, mais la petite application que vous allez créer offre une étude rapide du processus d'appel des routines API. En fait, de nombreuses fonctions et commandes de Visual Basic correspondent exactement à des appels API,et Microsoft appelle en interne la routine API nécessaire quand vous utilisez une telle fonction ou commande.

Créez un projet qui contient un unique bouton de commande au centre de la feuille. Le nom du bouton sera `cdmBeep` et son titre `&Bip`. Double-cliquez sur le bouton de commande pour ouvrir une procédure événementielle `Click`.

Lancez la Visionneuse d'API, si elle ne l'est pas déjà. Chargez le fichier win32api.txt (ou la base de données si vous l'avez convertie) et modifiez le type d'API pour afficher les déclarations. Dans la zone de texte, tapez `message` ; une liste de procédure API s'affichera, dont la première est `MessageBeep`. Modifiez l'option `Public` en `Private`, puis double-cliquez sur cette entrée pour afficher la déclaration de la procédure dans la zone de sélection.

Vous devez modifier le qualifiant `Public` en `Private`, car la déclaration de la fonction est locale au module feuille et n'est pas enregistrée dans une section de déclarations générales.

Sélectionnez toute la déclaration pour la coller dans votre fenêtre de code. Revenez dans la fenêtre de code de l'application Visual Basic. Insérez une ligne avant la procédure événementielle `Click` et collez la déclaration dans la section générale du module.

Ajoutez ensuite le code suivant dans la procédure événementielle `Click` du bouton de commande :

```
Dim Beeper As Variant
Beeper = MessageBeep(1)
```

Le Listing 23.1 montre la procédure complète.

Listing 23.1 : Vous pouvez utiliser l'API Windows pour déclencher le haut-parleur

```
1: Private Declare Function MessageBeep Lib "user32" (ByVal wType
   ➥As Long) As Long
2:
3: Private Sub cmdBeep_Click()
4:   Dim Beeper As Variant
5:   Beeper = MessageBeep(1)
6: End Sub
```

L'instruction `Declare` indique exactement à Visual Basic comment trouver la fonction `MessageBeep()`. Dans le cas d'une fonction interne de Visual Basic, telle que `Abs()`, il n'est pas nécessaire d'utiliser une instruction `Declare`, car il sait comment la trouver. La fonction `MessageBeep()`, quant à elle, se trouve hors de l'environnement de Visual Basic. L'instruction `Declare` est donc utilisée pour lui indiquer comment exécuter cette fonction et comment transmettre les valeurs. Elle signale en outre à Visual Basic que la fonction réside dans le fichier USER32.DLL.

Vous remarquerez que `MessageBeep()` est une fonction et pas une sous-routine. Elle renvoie donc une valeur, dont le type de données est Long. Votre application n'a cependant rien à faire de cette valeur. La variable Variant `Beeper` se contente d'enregistrer la valeur au retour de la fonction.

Quand vous exécutez le programme, le bouton de commande s'affiche au milieu de la feuille. Si vous cliquez dessus, vous entendrez un bip provenant de votre carte son. Si vous n'entendez rien, vous pouvez modifier l'argument en `-1` pour que le son provienne du haut-parleur interne du PC. Même sans carte son ou si les haut-parleurs de la carte sont éteints, le haut-parleur interne sonnera toujours.

Comme c'est le cas pour `Beep`, de nombreuses instructions Visual Basic dupliquent des routines API. C'est heureux, car cela évite d'avoir à appeler toutes les routines API que vous devriez sinon appeler. Il existe en outre plusieurs centaines de routines API de Windows dont vous n'aurez jamais besoin, car elles n'effectuent aucune tâche utile à un programme d'application.

Appel d'une API différente

Vous pouvez créer une autre application simple pour tester l'appel d'une routine de l'API Windows. Voyez le Listing 23.2 qui envoie dans une zone de texte le type de disque détecté.

Listing 23.2 : Utilisation de l'API Windows pour en savoir plus sur un disque dans votre application

```
 1:  Private Declare Function GetDriveType Lib "kernel32.dll"
     ➥Alias "GetDriveTypeA" (ByVal nDrive As String) As Long
 2:
 3:  Private Sub cmdDrive_Click()
 4:    Dim lngDriveType As Long
 5:
 6:    ' Transmettre le nom du disque qui vous intéresse
 7:    ' à la fonction GetDriveType()
 8:    lngType = GetDriveType("c:\")
 9:    '
10:   ' Utiliser la valeur renvoyée pour déterminer
11:   ' le type de disque testé
12:   Select Case lngType
13:     Case 2
14:       txtDrive.Text = "Disque amovible"
15:     Case 3
16:       txtDrive.Text = "Disque dur fixe"
17:     Case 4
18:       txtDrive.Text = "Disque distant (réseau)"
19:     Case Else
20:       txtDrive.Text = "Inconnu"
21:   End Select
22: End Sub
```

Pour saisir et tester le code du Listing 23.2, cherchez la déclaration `GetDriveType()` dans la Visionneuse d'API, changez l'option `Public` en `Private`, puis copiez-la dans le Presse-papiers. Complétez le reste du code pour qu'il corresponde au code du Listing 23.2.

Ajoutez ensuite un bouton de commande nommé `cmdDrive` en bas de la feuille, puis ajoutez un contrôle TextBox, nommé `txtDrive`, en haut de la feuille. Vous pouvez ajuster la police et la taille de ces contrôles à votre goût, mais ce n'est pas crucial dans le cadre de cet exemple. Modifiez la ligne 8 pour indiquer une autre unité, par exemple une disquette pour tester un autre type d'unité. Exécutez le programme pour voir s'afficher la description du disque (voir Figure 23.5).

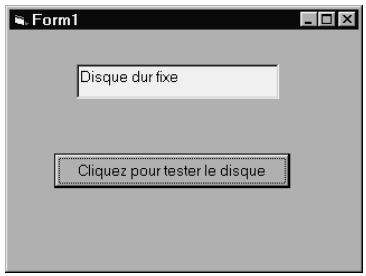


Figure 23.5
La routine API a renvoyé des informations sur votre unité de disque.

Trouver le dossier Windows

Parfois, lorsqu'ils écrivent des programmes qui accèdent à des fichiers système ou qui enregistrent des fichiers dans le répertoire Windows, les programmeurs Visual Basic utilisent les routines API pour localiser le dossier Windows (dans lequel il est installé). Vous pouvez aussi avoir à localiser les dossiers System et Temp. Il existe des routines API qui donnent ces informations. Vous pouvez, par exemple, enregistrer des fichiers temporaires de votre application dans le dossier Temp de l'utilisateur. (Le terme *dossier* est un synonyme de *répertoire*).

Supprimez toujours les fichiers que vous stockez dans le dossier Temp lors de l'exécution de votre application. De nombreux utilisateurs suppriment régulièrement les fichiers du dossier Temp vieux de plus d'un jour ou deux pour le nettoyer et faire de la place. Il n'est pas souhaitable que votre application soit de celles qui enregistrent leurs fichiers dans Temp sans les supprimer, car ces utilisateurs risquent de ne pas apprécier vos programmes longtemps. Quant à ceux qui ne pensent pas à nettoyer leur dossier Temp (la grande majorité, dont beaucoup ignorent même l'existence), ils ne sauront même pas que leur espace disque diminue chaque fois que votre application se termine. C'est du gâchis.

Les étapes suivantes décrivent comment créer une petite application qui vous permet d'expérimenter la recherche d'informations sur le dossier de Windows :

1. Créez une nouvelle application.
2. Placez les contrôles du Tableau 23.2 dans la feuille. La Figure 23.6 montre le résultat que vous obtiendrez.

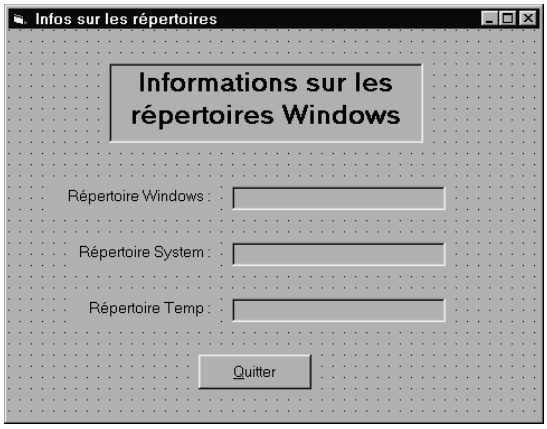


Figure 23.6
Cette feuille affichera les informations sur les dossiers de Windows.

3. Double-cliquez sur la feuille pour créer la procédure événementielle `Form _Load()`. Insérez quelques lignes vierges avant pour vous permettre d'inclure les instructions `Declare` qui renverront les trois répertoires.
4. Démarrer la Visionneuse d'API si vous l'aviez fermée lors de la session précédente. Ouvrez le fichier texte ou base de données Win32api.
5. Sélectionnez les entrées `Declare` dans la zone de liste.
6. Cherchez l'entrée de déclaration de `GetWindowsDirectory()` dans la liste et double-cliquez dessus pour afficher la déclaration correspondante.
7. Changez l'option `Public` en `Private`, car la déclaration résidera dans le module de feuille, et pas dans un module de code général.
8. Copiez la fonction `GetWindowsDirectory()` dans la Visionneuse d'API et collez-la au début de la fenêtre de code.
9. Répétez les étapes 6 à 8 pour les déclarations des fonctions `GetSystemDirectory()` et `GetTempPath()`.
10. Entrez le reste du code nécessaire, décrit dans le Listing 23.3.
11. Exécutez l'application pour voir le résultat de la recherche d'informations sur les répertoires. La Figure 23.7 montre l'affichage résultant sur une machine.

Tableau 23.2 : Placez ces contrôles dans la feuille de votre application pour expérimenter la recherche des dossiers de Windows

Propriété de contrôle	*Valeur de propriété*
Feuille : `Name`	`frmFolder`
Feuille : `Caption`	`Infos sur les répertoires`
Feuille : `Height`	`4500`
Feuille : `Width`	`5790`
Label #1 `Name`	`lblTitle`
Label #1 `Alignment`	`2-Center`
Label #1 `BorderStyle`	`1-Fixed Single`
Label #1 `Caption`	`Informations sur les répertoires Windows`
Label #1 `FontSize`	`14`
Label #1 `FontStyle`	`Gras`
Label #1 `Height`	`855`
Label #1 `Left`	`1080`
Label #1 `Top`	`360`
Label #1 `Width`	`3375`
Label #2 `Name`	`lblWD`
Label #2 `Alignment`	`1-Right Justify`
Label #2 `Caption`	`Répertoire Windows :`
Label #2 `Height`	`255`
Label #2 `Left`	`720`
Label #2 `Top`	`1680`
Label #2 `Width`	`1455`
Label #3 `Name`	`lblSD`
Label #3 `Alignment`	`1-Right Justify`
Label #3 `Caption`	`Répertoire System :`
Label #3 `Height`	`255`
Label #3 `Left`	`720`
Label #3 `Top`	`2280`

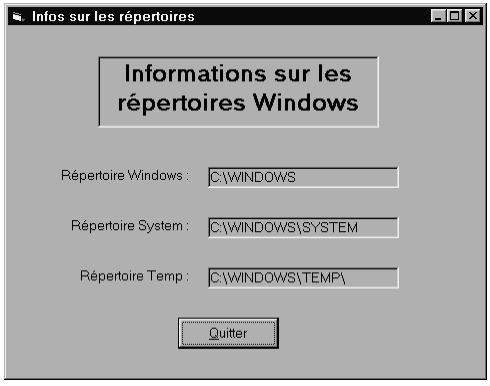
Tableau 23.2 : Placez ces contrôles dans la feuille de votre application pour expérimenter la recherche des dossiers de Windows *(suite)*

Propriété de contrôle	*Valeur de propriété*
Label #3 `Width`	`1455`
Label #4 `Name`	`lblTD`
Label #4 `Alignment`	`1-Right Justify`
Label #4 `Caption`	`Répertoire Temp :`
Label #4 `Height`	`255`
Label #4 `Left`	`720`
Label #4 `Top`	`2880`
Label #4 `Width`	`1455`
Label #5 `Name`	`lblWinD`
Label #5 `Alignment`	`0-Left Justify`
Label #5 `BorderStyle`	`1-Fixed Single`
Label #5 `Height`	`255`
Label #5 `Left`	`2400`
Label #5 `Top`	`1680`
Label #5 `Width`	`2295`
Label #6 `Name`	`lblWinS`
Label #6 `Alignment`	`0-Left Justify`
Label #6 `BorderStyle`	`1-Fixed Single`
Label #6 `Height`	`255`
Label #6 `Left`	`2400`
Label #6 `Top`	`2280`
Label #6 `Width`	`2295`
Label #7 `Name`	`lblWinT`
Label #7 `Alignment`	`0-Left Justify`
Label #7 `BorderStyle`	`1-Fixed Single`
Label #7 `Height`	`255`

Tableau 23.2 : Placez ces contrôles dans la feuille de votre application pour expérimenter la recherche des dossiers de Windows *(suite)*

Propriété de contrôle	*Valeur de propriété*
Label #7 `Left`	`2400`
Label #7 `Top`	`2880`
Label #7 `Width`	`2295`
Bouton de commande : `Name`	`cmdExit`
Bouton de commande : `Caption`	`&Quitter`
Bouton de commande : `Left`	`2040`
Bouton de commande : `Top`	`3480`
Bouton de commande : `Width`	`1215`

Figure 23.7
Les informations sur les répertoires dépendent de la configuration de la machine de l'utilisateur.



Listing 23.3 : Les fonctions de l'API qui recherchent les dossiers demandent un peu plus de travail que les routines API déjà vues

```
1:  Private Declare Function GetWindowsDirectory Lib "kernel32" Alias
➥"GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long)
➥As Long
2:  Private Declare Function GetSystemDirectory Lib "kernel32" Alias
➥"GetSystemDirectoryA" (ByVal lpBuffer As String, ByVal nSize  As Long)
➥As Long
3:  Private Declare Function GetTempPath Lib "kernel32" Alias
    ➥"GetTempPathA" (ByVal nBufferLength As Long, ByVal lpBuffer
```

Listing 23.3 : Les fonctions de l'API qui recherchent les dossiers demandent un peu plus de travail que les routines API déjà vues *(suite)*

```
    ➥As String) As Long
 4: Private Sub Form_Load()
 5:   ' Initialise les labels des dossiers système au chargement
 6:   ' Déclare une chaîne fixe assez longue pour contenir les informations
 7:   Dim strFolder As String * 255
 8:   Dim intLength As Integer
 9:
10:   '
11:   ' Obtient les informations sur le répertoire Windows
12:   intLength = GetWindowsDirectory(strFolder, 255)
13:   lblWinD.Caption = Left(strFolder, intLength)
14:   '
15:   ' Obtient les informations sur le répertoire System
16:   intLength = GetSystemDirectory(strFolder, 255)
17:   lblWinS.Caption = Left(strFolder, intLength)
18:   '
19:   ' Obtient les informations sur le répertoire Temp
20:   intLength = GetTempPath(255, strFolder)
21:   lblWinT.Caption = Left(strFolder, intLength)
22: End Sub
23:
24: Private Sub cmdExit_Click()
25:   End
26: End Sub
```

Le code que vous devez utiliser pour rechercher les noms des dossiers est intéressant, car il demande plus de travail que les précédents appels d'API que vous avez étudiés. Ces routines recherchent les informations de dossiers nécessaires, mais vous devez les récupérer dans une longue chaîne de caractères renvoyée par la fonction.

La ligne 7 réserve de la place pour une chaîne de longueur fixe de 255 caractères. Même si les systèmes ne demandent pas tant de caractères pour les dossiers, il vaut mieux en avoir trop que pas assez. La ligne 8 déclare ensuite une variable entière qui sera utilisée comme valeur de retour de chaque fonction. Cette valeur, stockée dans `intLength` (ligne 12), contient le nombre de caractères effectifs du chemin de dossier dans la chaîne. Selon les ordinateurs, la longueur du nom de chemin peut être différente, en fonction de l'emplacement des dossiers. Une chaîne de 255 caractères laisse à votre application beaucoup de place pour contenir les chemins d'accès de ces dossiers sur n'importe quel PC, sauf dans le cas, très rare où un de ces noms de répertoires dépasse 255 caractères.

Lorsque le dossier est obtenu à la ligne 12, remarquez que la fonction de l'API `GetWindowsDirectory()` demande comme arguments une chaîne pour enregistrer le chemin d'accès et la longueur maximale de cette chaîne. Elle ne tentera donc pas de stocker plus de 255 caractères dans la chaîne `strFolder`.

Une fois que la fonction revient, la ligne 13 récupère cette partie gauche, qui contient le nom de chemin. La valeur de retour de la fonction de l'API détermine le nombre de caractères du chemin d'accès dans la chaîne. Tous les caractères après ce nombre contiennent des informations non significatives, qu'il n'est pas souhaitable d'afficher.

La ligne 20 montre une anomalie qui se rencontre souvent dans les routines API. La fonction `GetTempPath()` est du même type que `GetWindowsDirectory()` et `GetSystemDirectory()`, pourtant la position de ses arguments est inversée. Faites toujours très attention aux instructions `Declare` quand vous travaillez sur des routines API pour ne pas confondre les arguments par inadvertance.

La particularité des trois fonctions illustrées dans cette application s'avère assez courante. De nombreuses routines de l'API Windows exigent de telles manipulations avant et après les appels de fonction. Parfois, des fonctions identiques ont des arguments inversés (comme ici `GetTempPath()`). Les intérêts des fonctions internes de Visual Basic est qu'elles offrent plus de cohérence, qu'elles s'associent parfaitement aux types de données intégrés à Visual Basic et qu'elles n'exposent pas aux mêmes dangers en cas d'utilisation incorrecte (par exemple, une réinitialisation intempestive de la machine).

En résumé

Cette leçon supplémentaire a expliqué comment utiliser les routines de l'API Windows. Windows est en fait un regroupement de bibliothèques de liens dynamiques qui contiennent des milliers de routines auxquelles on peut accéder depuis un programme Visual Basic. Vous n'utiliserez pas toutes les procédures disponibles dans l'API, mais certaines sont pratiques lorsque vous devez travailler sur des informations du système ou effectuer une fonction apparentée, par exemple lire des données de la base de registres ou réinitialiser la machine de l'utilisateur.

La maîtrise de l'API Windows demande du temps, et nombre de programmeurs n'en connaissent pas toutes les procédures — beaucoup sont nécessaires au bon fonctionnement du système d'exploitation, mais n'ont aucune utilité pour les applications que l'utilisateur exécute. Cependant, en ayant accès à ces procédures de l'API Windows

(fonctions et sous-routines) vous pouvez piocher dans un assortiment riche qui informe ou aide à gérer le système de l'utilisateur.

L'utilisation de l'API Windows est assez lourde. Cependant, Visual Basic comprend un outil, la *Visionneuse d'API*. Vous pouvez l'utiliser pour consulter toutes les routines existant dans l'API Windows et récupérer l'instruction de déclaration correspondante, que vous pouvez coller directement dans votre propre code. Une fois collée avec l'instruction `Declare`, la routine de l'API Windows peut être appelée dans votre programme Visual Basic. Windows rendra la fonction disponible au moment de l'exécution.

Questions-réponses

Q Pourquoi les déclarations des routines de l'API Windows et l'appel des procédures semblent si complexes ?

R La complexité vient de la connexion entre Visual Basic et la routine elle-même. Les routines API n'ont pas été écrites en fonction de Visual Basic. Elles ont été conçues et écrites à l'origine pour être utilisées avec le langage C, qui se sert d'un ensemble de types de données légèrement différent de celui de Visual Basic, et qui appelle aussi les routines d'une manière différente. La déclaration est la manière de faire comprendre à Visual Basic des routines qui ne font pas partie de son environnement.

Q Quels sont les autres exemples de routines API que je peux vouloir étudier ?

R Les routines sont bien trop nombreuses pour être mentionnées. Cependant, la liste suivante, même si elle est loin d'être exhaustive, vous donne un point de départ pour savoir ce qui est à votre disposition :

- rechercher les valeurs de la base de registres ;
- déterminer les ressources système libres et utilisées, comme l'espace mémoire ou disque ;
- accéder à la version de Windows qui s'exécute ;
- travailler sur une fenêtre ;
- graphiques de bas niveau ;
- gérer les valeurs dans un fichier INI (les fichiers INI étaient utilisés par les versions de Windows avant Windows 95 pour enregistrer des informations système et sont encore utilisés par certains programmes).

Trouver une routine relève parfois de la devinette. En général, si vous trouvez une routine API que vous pensez utile, recherchez dans l'aide en ligne de Visual Basic pour voir si elle répond à vos besoins. Vous ne pouvez toujours pas déduire l'action d'une routine à partir de son nom. Par exemple, pour lire et écrire une valeur dans un fichier INI, vous utilisez les fonctions Visual Basic `GetSetting()` et `SaveSetting()`.

Q Comment puis-je découvrir quelles routines de l'API Windows sont disponibles ?

R Il existe plusieurs sources, que ce soit le système d'aide en ligne de Visual Basic, le site Web de Microsoft ou encore divers ouvrages traitant de la programmation Visual Basic.

Q Existe-t-il un moyen de simplifier l'utilisation des routines API ?

R Si vous utilisez souvent une ou plusieurs routines API, vous pouvez simplifier quelque peu leur utilisation. Au lieu de lancer la Visionneuse d'API chaque fois que votre application doit utiliser une routine API pour coller l'instruction `Declare`, puis appeler la routine, vous pouvez ajouter les routines API que vous utilisez régulièrement dans un module de code standard avec une extension de fichier .BAS. Le module contiendra toutes les instructions `Declare` des routines API que vous voulez utiliser. Vous pouvez aussi écrire une fonction ou une sous-routine Visual Basic qui appelle la routine API. Utilisez une liste d'arguments qui correspond, en type de données, à ceux de la routine API. Une telle procédure permet d'enrober, autour de la routine de l'API Windows, une procédure Visual Basic qui en respecte les conventions d'appel et les arguments.

Ensuite, lorsqu'une application Visual Basic a besoin de faire appel à l'une ou l'autre de ces routines, il suffit d'ajouter le module à votre application. Cette dernière n'a alors plus qu'à appeler la fonction d'enrobage Visual Basic pour exécuter la routine API. En d'autres termes, pour votre application, une routine API est appelée et revient comme toutes les autres procédures Visual Basic que vous écrivez. Une fois que votre bibliothèque Visual Basic des routines API est correctement déboguée, vous disposez d'une méthode plus sûre d'appel des procédures API dont vous avez le plus besoin.

Atelier

L'atelier propose une série de questions qui vous aident à renforcer votre compréhension des éléments traités, ainsi que des exercices qui vous permettent de mettre en pratique ce que vous avez appris. Essayez de comprendre les questions et les exercices avant de passer à la leçon suivante. Les réponses se situent à l'Annexe A.

Quiz

1. Quelle est la signification d'*API* ?
2. Etant donné la riche collection des fonctions internes de Visual Basic, pourquoi pouvez-vous avoir besoin d'appeler une routine de l'API Windows ?
3. Pourquoi les DLL utilisées par vos applications Visual Basic n'augmentent pas la taille de ces dernières ?
4. Pourquoi les noms des DLL standards ont-ils changé avec les versions de Windows ?
5. Quel outil vous permet de consulter plus facilement le format des routines API ?
6. Quelle est l'instruction qui permet de déclarer les routines de l'API Windows ?
7. Vrai ou faux. Les routines de l'API Windows ont une apparence et un mécanisme d'appel uniforme.
8. Que fait l'instruction `Declare` ?
9. Quel qualifiant, `Public` ou `Private`, est nécessaire pour une procédure de l'API Windows déclarée dans un module de feuille ?
10. Quel est le but d'une procédure d'enrobage ?

Exercice

1. Quel fichier, GDI32.DLL ou KERNEL32.DLL, contient la fonction de l'API `GetSystemTime()` ? Comment pouvez-vous le savoir de toute routine API que vous rencontrez ?

Partie IV

Annexes

A

Solutions aux exercices

Chapitre 1

Quiz

1. Visual Basic est fondé sur le langage BASIC.
2. Visual Basic est un environnement visuel et exploite un langage simple, dérivé du BASIC. Mais il permet de créer de puissantes applications Windows.
3. La nature visuelle de Visual Basic prime sur le langage de programmation à plusieurs égards. L'interface visuelle donne à vos programmes leur "visage" particulier et interagit avec l'utilisateur. Le langage de programmation travaille en coulisses pour connecter l'ensemble des éléments visuels.
4. Une fenêtre de feuille peut être la fenêtre d'application, mais une même application peut contenir plusieurs fenêtres de feuilles. La fenêtre de feuille est ce voit l'utilisateur quand il exécute le programme.
5. Un bogue est une erreur dans le programme. Le débogage est le processus de correction des bogues.
6. Les programmes écrits en langage compilé s'exécutent beaucoup plus rapidement que les programmes écrits en langage interprété.
7. Les programmes écrits en langage interprété sont plus faciles à déboguer que les programmes écrits en langage compilé.

8. L'ecran de présentation s'affiche, toujours identique, au démarrage de l'application. L'écran Astuce du jour n'apparaît qu'après, sert à donner des conseils à l'utilisateur, et cet utilisateur peut demander que cet écran ne s'affiche plus.
9. Les contrôles sont des objets tels que labels, boutons de commande, boutons d'option, etc., qui apparaissent sur la feuille. Les propriétés et valeurs de propriétés d'un contrôle sont ce qui définit son comportement et le distingue des autres.
10. Faux. Les contrôles ne contiennent pas de code. Les contrôles sont des objets visuels avec lesquels l'utilisateur interagit. Le code est stocké séparément (dans ce que l'on appelle un module).

Exercice

Lancez l'assistant Création d'applications et répondez Oui lorsqu'il demande si vous souhaitez ajouter un accès Internet à votre application. Lors de l'exécution, vous sélectionnerez Affichage, Navigateur Web pour ouvrir la fenêtre navigateur. (Vous devez naturellement disposer d'une connexion Internet.)

Chapitre 2

Quiz

1. Une barre d'outils est un ensemble de commandes accessibles par simples boutons. La Boîte à outils est la fenêtre qui contient les contrôles que vous pouvez placer sur la feuille.
2. MSDN.
3. Faux. La fenêtre Feuilles peut contenir autant de feuilles que l'application l'exige. Toutefois, une seule feuille peut être sélectionnée à la fois.
4. Lorsque vous le placez sur la feuille après avoir sélectionné un outil, le pointeur flèche de la souris se transforme en pointeur cruciforme qui permet de dessiner le contrôle.
5. Visual Basic place automatiquement le contrôle au centre de la feuille. Vous pouvez ensuite le déplaer et le dimensionner.
6. Faux. C'est dans la fenêtre Propriétés que l'on définit les propriétés de contrôles (on peut également le faire dans le code).
7. La fenêtre Propriétés affiche les propriétés de l'objet sélectionné.

8. En cliquant sur les points de suspension, par exemple à la propriété `Font`, vous affichez une boîte de dialogue qui facilite le réglage de la propriété.
9. `Caption`.
10. Les noms par défaut ne sont pas très explicites. Attribuer au contrôle un nom plus parlant, notamment à l'aide de préfixes spécifiques, cela permet de deviner sa fonction d'après le seul nom, ce qui est pratique lorsqu'un projet contient de nombreux contrôles.

Exercices

Pour ajouter un arrière-plan bleu, double-cliquez sur la propriété `BackColor` de la feuille. (La desription donnée en bas de la fenêtre Propriétés indique la fonction de la propriété sélectionnée.) Une zone de liste s'affiche, qui contient deux onglets. A l'onglet Palette, vous pouvez choisir n'importe quelle couleur d'arrière-plan. L'onglet Système propose les couleurs Windows plus conventionnelles. A l'onglet Palette, sélectionnez un ton de bleu.

Ajouez un bouton de commande quelque part sur la feuille, par exemple dans le coin inférieur droit. Changez sa propriété `Name` en `cmdExit`. Changez sa propriété `Caption` en `Quitter`. Pour ajouter la ligne de code requise, double-cliquez sur le bouton de commande. Tapez `End` au milieu de la procédure. Exécutez le programme. Le nouveau bouton de commande apparaît sur fond bleu. Une fois l'image affichée, vous pouvez quitter le programme en cliquant sur Quitter.

Chapitre 3

Quiz

1. Une touche de raccourci est une combinaison de touche, par exemple Alt-R, par laquelle l'utilisateur peut sélectionner le contrôle.
2. Faux. Les événements concernent uniquement les contrôles.
3. L'événement `Cancel` permet à l'utilisateur de sélectionner le bouton de commande en appuyant sur la touche Echap.
4. Le contrôle qui a le focus est entouré d'un cadre en pointillé.
5. En appuyant sur Tab ou Maj-Tab (et sur la flèches dans certaines applications).
6. `TabIndex`.

7. les parenthèses indiquent que `LoadPicture ()` est une sorte de procédure. `LoadPicture ()` est la procédure qui avait permis d'afficher l'image, au Chapitre précédent. `LoadPicture ()` est une fonction interne : elle ne nécessite pas de code, car le code réside déjà dans le langage de programmation Visual Basic. Les fonctions internes ont ceci de pratique qu'elle vous épargne de saisir les codes appelés à servir souvent. Vous découvrirez ces fonctions internes au Chapitre 5.

8. Faux. Visual Basic génère souvent une procédure événementielle `Click`, mais pas toujours. Visual Basic insère la première et la dernière ligne de la procédure événementielle la plus probable pour le contrôle. Dans le cas d'une zone de texte, par exemple, ce sera la procédure événementielle `Change`.

9. Vrai.

10. La propriété `PasswordChar` permet de masquer ce que tape l'utilisateur. Les regards indiscrets, ainsi, ne peuvent attraper par-dessus l'épaule un mot de passe ou autre information sensible.

Exercices

1. `1: Private Sub frmMyApp_Load ()`

2. Les lignes 1 et 5 indiquent que la procédure est une fonction. Les procédures événementielles sont des sous-routines. Remplacez `Function` par `Sub` pour résoudre le problème.

3. Les trois zones de texte doivent avoir des propriétés `Style` différentes pour les trois barres de défilement. La propriété `Text` doit être définie lors de la création. Vous ne pouvez spécifier plusieurs lignes dans `Text`. Un mot ou une phrase suffiront.

   Définissez comme `&Quitter` la propriété `Caption` du bouton de commande.

Chapitre 4

Quiz

1. Le Créateur de menus.

2. Vrai.

3. `Click` (c'est d'ailleurs le seul).

4. Un raccourci clavier est une combinaison de touches (Ctrl-B, Alt-A, etc.) permettant d'exécuter une commande.
5. Les raccourcis clavier permettent de sélectionner rapidement les options de menu.
6. `Click`.
7. Faux. Le Créateur de menus ne sert qu'à concevoir la structure du menu, non le code.
8. Une coche s'affiche à gauche de l'option de menu.
9. Vrai, par défaut. C'est dans le code que vous spécifiez le contraire.
10. Ce sont des commentaires, qui décrivent le programme en langage clair. Le Projet bonus 1, situé à la fin du Chapitre 4, vous en dira plus.

Exercices

1. Le Créateur de menus contient une série de boutons figurant des flèches. Lorsqu'on clique sur la flèche-droite, des points de suspension s'insèrent, et l'élément sélectionné descend d'un cran dans la hierarchie du menu : option de menu, sous-menu, etc.
2. Manuel doit comprendre que c'est à lui de spécifier, dans le code, qu'une option de menu doit être désactivée. Les options de menu ne sont pas toujours mutuellement exclusives, comme elles l'étaient dans ce chapitre. C'est au programmeur de définir le comportement des coches.
3. Ouvrez la fenêtre Code et cliquez sur chaque option de menu dans la aprtie inférieure de la fenêtre. Sélectionnez des raccourcis clavier dans la liste fournie par le Créateur de menus. A l'exécution du programme, les raccourcis clavier apparaîtront vis-à-vis des options de menu correspondantes. Appuyez sur l'une des combinaisons de touche pour faire un essai.

Chapitre 5

Quiz

1. Les déclarations de variables et autres données.
2. Une variable locale peut être partagée par plusieurs procédures, comme vous allez le découvrir dans les chapitres à venir.
3. Vrai.

4. Faux. Une variable peut prendre successivement plusieurs valeurs lors de l'exécution du programme.
5. L'un effectue une division classique, l'autre une division entière.
6. Un opérateur surchargé est un opérateur qui peut effectuer des opérations différentes selon le contexte.
7. L'esperluette (`&`). Le signe plus (`+`) doit être de préférence réservé aux additions.
8. `Variant`.
9. Faux. Les préfixes ne sont pas obligatoires, mais facilitent la documentation du code.
10. Vous pouvez spécifier l'instruction `Option Explicit` au début de la section de déclarations ou cochez l'option correspondante dans la boîte de dialogue Options. La boîte de dialogue s'applique globalement, alors que l'instruction `Option Explicit` ne concerne que son propre module de code.

Exercices

1. Vous pensez que la variable `abc` provoquera une erreur ? Bien vu, mais ce n'est pas ça. A moins que vous ne spécifiez autre chose, Visual Basic traite a priori toutes les variables comme de type `Variant`. La variable `abc` sera donc déclarée comme `Variant`.
2. Visual Basic calcule la division avant l'addition, mais pour obtenir la moyenne, Marie doit procéder ainsi : `sngAvg = (sngGrade1 + sngGrade2 + sngGrade3) / 3`
3. a. 5.

   b. 6.

   c. 5.

   d. 5.

   e. 7.
4. a. `a = (3 + 3) / (4 + 4)`

   b. `x = (a - b) * (a - 2) ^ 2`

   c. `f = a ^ (1/2) / b ^ (1/2)`
5. Les esperluettes concatènent les chaînes de sorte que Visual Basic traite les multiples lignes comme un seul littéral.

Chapitre 6

Quiz

1. `Or`.
2. L'opérateur conditionnel compare deux valeurs. L'opérateur logique combine deux expressions conditionelles.
3. Une série d'instructions appelées à se répéter plusieurs fois.
4. Après cette affectation, la variable de type `Integer` contiendra 10 unités de moins qu'avant.
5. Cette boucle ne s'exécutera jamais, car `intN` est définie comme égale à zéro.
6. Si `Exit For` n'était pas inclus dans une instruction `If`, la boucle `For` ne s'exécuterait qu'une seule fois, puisque `Exit For` provoquerait l'arrêt de la boucle dès la première itération.
7. Faux. Le bloc `If` block s'exécute si la condition est satisfaite, le bloc `Else` si elle ne l'est pas.
8. Vrai. Si, au commencement de la boucle, la valeur finale est déjà plus grande que la valeur initiale, la boucle ne s'exécutera jamais.
9. Pour exécuter une boucle plusieurs fois.
10. Selon son résultat, une instruction de décision peut n'exécuter le code qu'une seule fois. Une instruction de boucle exécute son code plusieurs fois.

Exercices

1. `If (a = b) And (b = c) Then ' Le corps de l'instruction suit.`
2. La boucle du pauvre Momo n'a pas de fin, parce que la variable de contrôle ne change jamais de valeur à l'intérieur de la boucle.
3. Dans la réalité, l'horloge d'un stade de foot n'effectue pas un *décompte*, mais l'exemple illustre le fonctionnement des boucles imbriquées :

```
1:  For Qtr = 1 to 2
2:    For Minutes = 45 to 0 Step -1
3:      ' Mi-temps.
4:    Next Minutes
5:  Next Qtr
```

La boucle intérieure effectue un décompte de 45 à 0, et la boucle extérieure répète ce décompte deux fois.

4. Le code suivant explote trois types de `Case` dans une même instruction `Select Case` :

```
1:  Select Case intHours
2:    Case 1 To 40
3:      curOverTime = 0.0
4:    Case 41 To 49
5:      curOverTime = (intHours - 40) * 1.5 * sngRate
6:    Case Is >= 50
7:      curOverTime = ((intHours - 50) * 2 + (10 *  1.5)) *
        ➥sngRate
8:  End Select
```

Chapitre 7

Quiz

1. La fonction interne renvoie une nouvelle valeur fondée sur la valeur de l'argument qui lui est passé.
2. Vrai.
3. `Empty`.
4. Faux. Les trois tableaux décrivent un même argument de la fonction `MsgBox()`, représenté dans le chapitre par *`intStyle`*.
5. Le nom du projet.
6. L'utilisateur peut cocher autant de cases à cocher que nécessaire, mais il ne peut sélectionner qu'un seul bouton d'option à la fois.
7. Vrai. Il faut pour cela que leurs valeurs soient `False` au premier chargement de la feuille (par exemple, dans la procédure événementielle `Form_Load()`).
8. Respectivement, les valeurs `0` et `1` de la propriété `Value`.
9. Respectivement, les valeurs `True` et `False` de la propriété `Value`.
10. Afin que l'utilisateur puisse en sélectionner plusieurs à la fois, répartis en catégories différentes.

Exercices

1. On vérifie si `InputBox()` renvoie une chaîne vide (`""`).
2.
```
strAns = MsgBox("You are out of paper", vbExclamation + _
vbDefaultButton2 + vbAbortRetryIgnore, "For Printer")
```
3. Voici la procédure événementielle :

```
1:  Private Sub cmdGetLocation_Click()
2:  ' Obtient la ville et le département par deux boîtes
    ➥d'entrée.
3:  ' Concatène les deux chaînes renvoyées.
4:  ' Affiche le résultat.
5:     Dim strCity As String   ' Ville.
6:     Dim strState As String  ' Département.
7:     Dim strBoth As String   ' Chaîne concaténée.
8:     ' Valeur de renvoi de MsgBox().
9:     Dim intAnswer As Integer
10:
11:     ' Demande la ville et le département.
12:     strCity = InputBox("Quelle sont la ville ?", "Ville")
13:     strState = InputBox("Quelle est le département ?",
        ➥"Département")
14:
15:     ' Concatène les chaînes.
16:     strBoth = strCity & ", " & strState
17:
18:     ' Affiche la chaîne finale.
19:     intAnswer = MsgBox("Vous habitez à " & strBoth,
        ➥"Résidence")
20: End Sub
```

4. Les procédures événementielles pour les boutons d'option seront des procédures événementielles `Click`. `Click` est l'événement qui se produit lorsque l'utilisateur sélectionne l'un des boutons d'option. Vous n'avez pas à désélectionner les autres, Visual Basic s'en charge automatiquement.

Chapitre 8

Quiz

1. Les variables publiques.
2. Les variables locales.
3. Vrai. Les arguments sont toujours passés par référence si vous ne spécifiez pas `ByVal`.

4. Les sous-routines ne renvoient pas de valeurs.
5. `IIf()` et `Choose()`.
6. `Choose()` renvoie alors `Null`.
7. `Abs()` renvoie la valeur absolue de son argument.
8. a. `74135-`

   b. `12345.67`
9. `intN` contiendra `192`.
10. `Now` renvoie la date et l'heure, tandis que `Time` ne renvoie que l'heure.

Exercices

1. Voici le Listing 8.1 réécrit :

```
1:  Private Sub GetTotal()
2:  ' Cette procédure additionne les valeurs d'un
3:  ' formulaire, puis envoie le total et le pourcentage
4:  ' d'abattement à la procédure qui calcule la taxe.
5:     Dim curTotal As Currency
6:     Dim sngDisc As Single   ' Special tax discount
7:     '
8:     ' Calculer le total d'après le formulaire.
9:     curTotal = txtSale1.Text + txtSale2.Text + txtSale3.txt
10:    '
11:    ' Envoyer le total à la procédure de taxation.
12:    intMsg = MsgBox("The sales tax is " & SalesTax(curTotal,
       ➥_sngDisc))
13: End Sub
14:
15: Public Function SalesTax(curTotal As Currency, sngRateDisc
    ➥As _Single) As Currency
11:   ' Envoyer le total à la procédure de taxation.
17:   Dim curSalesTax As Currency
18:   '
19:   ' Dans le chapitre, ce code était une sous-routine.
20:   ' Calcul de la taxe de vente
21:   ' à 3,5 % du total.
22:   curSalesTax = (curTotal * .03) + (curTotal * .005)
23:   '
24:   ' Déduction du pourcentage d'abattement.
25:   curSalesTax = curSalesTax - (sngRateDisc * curTotal)
26:   '
```

```
27:    ' Définit la valeur renvoyée.
28:    SalesTax = curSalesTax
29:    '
30:    ' Une fois terminée, la procédure revient à
31:    ' la procédure appelante.
32: End Function
```

2. `strTitle = IIf(intTotal > 1000, "Bon boulot !", "Viré !")`
3. `Choose(ID, intBonus = 50, intBonus = 75, intBonus = 100)`
4. `intN contient -6, intO contient -5, et intP contient -6`

Chapitre 9

Quiz

1. Il faut ajouter le contrôle depuis la boîte de dialogue Composants.
2. Ouvrir, Enregistrer, Police, Couleur, Imprimer et Aide Windows.
3. Le contrôle Common Dialog permet de générer six boîtes de dialogue standards.
4. Le contrôle Common Dialog ne devient visible pour l'utilisateur que lorsqu'on lui applique l'une de ses méthodes `Show`. La boîte de dialogue commune correspondante s'affiche alors au milieu de l'écran.
5. Vrai.
6. La propriété `Filter` détermine les types de fichiers qui pourront s'afficher.
7. La propriété `Flags` définit les divers paramètres d'une boîte de dialogue commune, avant qu'elle ne s'affiche.
8. Vrai.
9. Faux. La boîte de dialogue Imprimer ne requiert pas de valeurs `Flags` prédéfinies pour s'afficher. Cependant, on règle en général certaines propriétés, telles que `DialogTitle`, avant l'affichage.
10. Faux. La méthode doit être plus précise : `ShowFont`, `ShowPrinter`, etc.

Exercices

1. Voici le Listing 9.2 modifié :

```
1:   ' Présuppose que CancelError vaut True.
2:   On Error Goto dbErrHandler
3:   ' Définit les valeurs Flags.
4:   CdbFont.Flags = cdlCFBoth Or cdlCFEffects
5:   CdbFont.ShowFont  ' Affiche la boîte de dialogue Police.
6:  ' Définit les propriétés du label qui
7:  ' reflètera les choix de l'utilisateur.
8:   LblMessage.Font.Name = CdbFont.FontName
9:   LblMessage.Font.Size = CdbFont.FontSize
10:  LblMessage.Font.Bold = CdbFont.FontBold
11:  LblMessage.Font.Italic = CdbFont.FontItalic
12:  LblMessage.Font.Underline = CdbFont.FontUnderline
13:  LblMessage.FontStrikethru = CdbFont.FontStrikethru
14:  LblMessage.ForeColor = CdbFont.Color
15:  Exit Sub
16:
17: dbErrHandler:
18:  ' L'utilisateur a cliqué sur Annuler.
19:  Exit Sub  ' Pas de modification.
```

2. Voici la procédure :

```
1:  Private Sub mnuFileOpen_Click ()
2:  ' Présuppose que CancelError vaut True.
3:    On Error Goto dbErrHandler
4:    ' Determine les types de fichiers
5:    ' qui apparaîtront.
6:    cdbFile.Filter = "Texte (*.txt) | *.txt"
7:    ' Spécifie le filtre par défaut.
8:    cdbFile.FilterIndex = 1
9:    cdbFile.DialogTitle = "Open"
10:   ' Affiche la boîte de dialogue Ouvrir.
11:   cdbFile.ShowOpen
12:
13:   '*********************************
14:   ' Ici, placez ou appelez une      *
15:   ' procédure qui ouvre le fichier  *
16:   ' sélectionné par l'utilisateur.  *
17:   '*********************************
18:   Exit Sub
19:
20: dbErrHandler:
21:   ' L'utilisateur a cliqué sur Annuler.
22:   Exit Sub  ' Ne pas ouvrir de fichier.
23: End Sub
```

Chapitre 10

Quiz

1. `MouseDown` et `MouseUp` répondent à des boutons spécifiques et indiquent, par les arguments qu'ils passent à leurs procédures, lequel des deux boutons de souris a servi. `Click`, `DblClick` et `MouseMove` ne tiennent pas compte du bouton qui a servi.
2. Par l'argument *`intButton`*.
3. Il suffit d'affecter le chemin d'accès d'une icône à la propriété `DragIcon` du contrôle.
4. En écrivant un code qui ignore certains événements timer et ne réponde qu'une fois que le laps de temps voulu s'est écoulé.
5. On peut initialiser le contrôle lors de la création en ajoutant les éléments dans la propriété `List`, ou lors de l'exécution (pratique la plus courante) en passant par la méthode `AddItem`.
6. La propriété `Value` du contrôle de liste spécifie l'indice de l'élément sélectionné.
7. On peut supprimer les éléments un à un avec la méthode `RemoveItem`, ou les supprimer tous d'un coup avec la méthode `Clear`.
8. Pour que la procédure événementielle puisse tenir compte du nouvel élément, il faut que le focus se déplace.
9. En définissant la propriété `Sorted` comme `True`.
10. 23 éléments.

Exercices

1. Ajoutez à votre feuille une zone de liste, que vous initialiserez avec le code suivant (aux prénoms près, naturellement) dans la procédure événementielle `Form_Load()`:

```
Private Sub Form_Load()
   lstFamily.AddItem ("Josette")
   lstFamily.AddItem ("Paulette")
   lstFamily.AddItem ("Mauricette")
   lstFamily.AddItem ("Idulphe")
   lstFamily.AddItem ("Tertullien")
   lstFamily.AddItem ("Yolande")
   lstFamily.AddItem ("Bruno")
   lstFamily.AddItem ("Elvis")
```

```
    lstFamily.AddItem ("Georg Wilhelm Friedrich")
    lstFamily.AddItem ("Marie-Chantal")
    lstFamily.AddItem ("Casimir")
End Sub
```

2. (Aucune réponse nécessaire.)
3. Une fois les trois ComboBox ajoutées, il faut écrire pour chacune d'elle une procédure événementielle `Change`. Lorsqu'une ComboBox est change, il faut affecter l'élément à la valeur d'index (l'élément sélectionné et ajouté) à la fin des autres ComboBox, en employant la méthode `AddItem` et en indexant l'élément dans la `ListCount` de ces contrôles.

Chapitre 11

Quiz

1. Lorsque l'événement `Resize` se produit, vous pouvez ajuster la position des contrôles aux nouvelles dimensions de la feuille.
2. La propriété `ScaleMode` détermine les unités de mesures qui seront employées pour les coordonnées `CurrentX` et `CurrentY`.
3. Le premier indice est `0`.
4. Faux. Les applications SDI peuvent supporter plusieurs feuilles, à condition que ces feuilles ne contiennent pas chacune un jeu de données différent.
5. Une application MDI peut contenir plusieurs fenêtres de feuilles, chacune contenant un jeu de données différent.
6. En haut.
7. ImageList.
8. Le contrôle ImageList ne fait que contenir les icônes des boutons. Il n'apparaît pas à l'exécution.
9. `Spc()` insère dans une ligne de sortie un nombre fixe d'espaces, tandis que `Tab()` positionne le curseur texte sur un colonne spécifique.
10. En utilisant la méthode `Print` toute seule.

Exercices

1. (Aucune réponse nécessaire.)
2. Les deux sorties `Print` s'affichent sur une seule ligne, à cause du point-virgule :

   ```
   Ligne 1Ligne 2
   ```

3. (Aucune réponse nécessaire.)
4. Votre procédure événementielle `Click` devrait contenir une instruction `For` comme celle-ci :

   ```
   For intCtr = 1 To 100
      Form1.Print intCtr; " ";
   Next intCtr
   ```

5. Voici le code :

   ```
   ' Commence à zéro.
   intCount = 0
   ' Passe par chaque feuille.
   For intCtr = 1 to Forms.Count
      ' Ajoute le nombre de contrôles sur chaque feuille.
      intCount = intCount + Forms(intCtr).Count
   Next intCtr
   ```

Chapitre 12

Quiz

1. Tous.
2. `FreeFile()`.
3. Visual Basic l'écrase.
4. Visual Basic écrira à la fin du fichier séquentiel à la prochaine instruction de sortie fichier.
5. l'instruction ouvre un fichier séquentiel et permet aux instructions de sorties subséquentes d'écrire à la fin du fichier.
6. Le fichiers séquentiels doivent être en mesure de localiser chaque numéro d'enregistrement. Pour que ce calucl fonctionne, les enregistrements doivent être de longueur identique.

7. Les chaînes de longueur fixe donnent des enregistrements de longueur fixe, ce qui n'est évidemment pas le cas des chaînes de longueur variable.

8. `Type`.

9. Faux. L'instruction définit un nouveau type de données, mais ne déclare aucune variable de ce type.

10. `Dir()` avec arguments renvoie le premier fichier trouvé qui corresponde aux jokers. `Dir()` sans arguments renvoie le prochain fichier du dossier qui corresponde aux jokers.

Exercices

1. Mauricette essaie sans doute de supprimer le dossier Factures qui contient encore des fichiers. La commande `RmDir` ne s'applique qu'aux dossiers vides.

2. La procédure suivante remplit le tableau (lignes 10 à 14) et écrit ces éléments dans le fichier ouvert (lignes 21 à 23) :

```
1:  Private Sub output ()
2:    Dim strNames(5) As String
3:    Dim intAges(5) As Integer
4:    Dim strColors(5) As String
5:
6:    Dim intCtr As Integer   ' Compteur de boucle.
7:    Dim intFNum As Integer  ' Numéro de fichier.
8:
9:    ' Reçoit les informations du tableau.
10:   For intCtr = 0 To 4
11:     strNames(intCtr) = InputBox("Nom suivant ?", "Nom")
12:     intAges(intCtr) = InputBox("Age suivant ?", "Age")
13:     strColors(intCtr) = InputBox("Couleur suivante ?", "Couleur")
14:    Next intCtr
15:
16:    intFNum = FreeFile
17:
18:    ' Ecrit la sortie.
19:    ' (Adaptez le chemin et le nom du fichier.)
20:   Open "C:\Stuff.txt" For Output As #intFNum
21:    For intCtr = 0 To 4
22:      Write #intFNum, strNames(intCtr), intAges(intCtr), _
         ➥strColors(intCtr)
23:    Next intCtr
24:
25:    Close #intFNum
26:  End Sub
```

3. Commencez par reproduire la feuille de la Figure 12.1. Pour chacune des trois zones de liste, vous devez écrire une procédure événementielle `Change` qui mette à jour les deux autres. Voici un exemple :

```
Private Sub Direct_Change()
  ' L'utilisateur a changé la liste Dossier.
  filFile.Path = dirList.Path  ' Change la liste Fichier.
  '
  ' Garantit qu'un seul fichier est sélectionné.
  If (filFile.ListCount > 0) Then
    ' Sélectionne le premier fichier de la liste.
    filFile.ListIndex = 0
  End If
End Sub
```

Lorsque l'utilisateur sélectionne un lecteur différent, vous devez en outre affecter le nouveau chemin vers ce lecteur :

```
Private Sub Drive_Change()
  ' Définit le chemin par défaut du lecteur.
  dirList.Path = drvList.Drive
End Sub
```

Chapitre 13

Quiz

1. En interrogeant la valeur `Printers.Count`.
2. Vrai.
3. `ScaleMode`.
4. Par la méthode `NewPage`.
5. `Is TypeOf`.
6. Vrai.
7. Faux. `KillDoc` ne peut annuler que les sorties imprimante de l'application.
8. `Me`.
9. A la résolution de l'écran.
10. `True`.

Exercices

1. `Printer.Print Tab(32); "Votre nom"`
2. Yolande doit comprendre que Windows supporte de multiples polices et tailles de police. Ses calculs quant à la largeur et à la fin de la page doivent tenir compte du corps de la police utilisée.
3. Voici le Listing 13.2 modifié :

```
1:  Public Function IsColor() As Boolean
2:    Dim blnIsColor As Boolean
3:    Dim prnPrntr As Printer
4:    '
5:    ' Présuppose qu'aucune imprimante couleur n'a été encore
   ➥trouvée.
6:    blnIsColor = False
7:    '
8:    ' Parcourt les imprimantes.
9:    For Each prnPrntr In Printers
10:     If prnPrntr.ColorMode = vbPRCMColor Then
11:       ' Définit l'imprimante couleur comme imprimante par
       ➥défaut.
12:       Set Printer = prnPrntr
13:       blnIsColor = True
14:       Exit For  ' Laisse tomber.
15:     End If
16:   Next   ' Parcourt les imprimantes si nécessaire.
17:   '
18:   ' blnIsColor reste False si aucune imprimante couleur
19:   ' n'est trouvée, et devient True dans le cas contraire.
20:   ' Définit en conséquence la valeur renvoyée par la
   ➥fonction.
21:   IsColor = blnIsColor
22: End Function
```

Chapitre 14

Quiz

1. Le contrôle Shape.
2. Le contrôle Shape (qui dessine également des rectangles).
3. `PSet` et `Ligne`.
4. Vrai.

5. L'option `F`.
6. Parce que les boutons ne sont pas tous nécessaires.
7. `Mode` renvoie l'état du périphérique du contrôle multimédia.
8. En réduisant la valeur de la propriété `UpdateInterval`.
9. C'est le contrôle PictureBox qui recevra la sortie vidéo.
10. C'est le code qui définit ainsi le contexte de périphérique.

Exercices

1. Voici à quoi devrait ressembler votre code :

```
1:  Private Sub Form_Load()
2:     ' Initialise la liste Forme.
3:     lstShape.AddItem "0 - Rectangle"
4:     lstShape.AddItem "1 - Square"
5:     lstShape.AddItem "2 - Oval"
6:     lstShape.AddItem "3 - Circle"
7:     lstShape.AddItem "4 - Rounded Rectangle"
8:     lstShape.AddItem "5 - Rounded Square"
9:
10:    ' Initialise la liste Motif.
11:    lstPattern.AddItem "0 - Solid"
12:    lstPattern.AddItem "1 - Transparent"
13:    lstPattern.AddItem "2 - Horizontal Ligne"
14:    lstPattern.AddItem "3 - Vertical Ligne"
15:    lstPattern.AddItem "4 - Upward Diagonal"
16:    lstPattern.AddItem "5 - Downward Diagonal"
17:    lstPattern.AddItem "6 - Cross"
18:    lstPattern.AddItem "7 - Diagonal Cross"
19:
20:    ' Initialise la liste Couleur.
21:    ' (Les couleurs ne sont pas toutes représentées.)
22:    lstFillColor.AddItem "Black"
23:    lstFillColor.AddItem "White"
24:    lstFillColor.AddItem "Blue"
25:    lstFillColor.AddItem "Red"
26:    lstFillColor.AddItem "Green"
27:    lstFillColor.AddItem "Yellow"
28:
29:    ' Initialise la liste Bordure.
30:    lstBorderColor.AddItem "Black"
31:    lstBorderColor.AddItem "White"
32:    lstBorderColor.AddItem "Blue"
33:    lstBorderColor.AddItem "Red"
34:    lstBorderColor.AddItem "Green"
35:    lstBorderColor.AddItem "Yellow"
```

```
36:
37:   ' Défininit comme valeur par défaut la première de chaque
      ➥liste.
38:   lstShape.ListIndex = 0
39:   lstPattern.ListIndex = 0
40:   lstFillColor.ListIndex = 0
41:   lstBorderColor.ListIndex = 0
42: End Sub
43:
44: Private Sub lstPattern_Click()
45:   ' Applique le motif sélectionné.
46:   shpSample.FillStyle = lstPattern.ListIndex
47: End Sub
48:
49: Private Sub lstShape_Click()
50:   ' Applique la forme sélectionnée.
51:   shpSample.Shape = lstShape.ListIndex
52: End Sub
53:
54: Private Sub lstFillColor_Click()
55:   ' Applique la couleur de fond sélectionnée.
56:   Select Case lstFillColor.ListIndex
57:     Case 0:
58:        shpSample.FillColor = vbBlack
59:     Case 1:
60:        shpSample.FillColor = vbWhite
61:     Case 2:
62:        shpSample.FillColor = vbBlue
63:     Case 3:
64:        shpSample.FillColor = vbRed
65:     Case 4:
66:         shpSample.FillColor = vbGreen
67:     Case 5:
68:         shpSample.FillColor = vbYellow
69:     End Select
70: End Sub
71:
72: Private Sub lstBorderColor_Click()
73:   ' Applique la couleur de bordure sélectionnée.
74:   Select Case lstBorderColor.ListIndex
75:   Case 0:
76:      shpSample.BorderColor = vbBlack
77:   Case 1:
78:      shpSample.BorderColor = vbWhite
79:   Case 2:
80:     shpSample.BorderColor = vbBlue
81:   Case 3:
82:      shpSample.BorderColor = vbRed
83:   Case 4:
84:       shpSample.BorderColor = vbGreen
85:   Case 5:
86:       shpSample.BorderColor = vbYellow
```

```
87:   End Select
88: End Sub
89:
90: Private Sub mnuFileExit_Click()
91:   End
92: End Sub
```

2. (Aucune réponse nécessaire.)
3. Utilisez le contrôle Common Dialog. Ajoutez une option de menu Fichier, Ouvrir qui affichera la boîte de dialogue Ouvrir. Définissez un filtre qui ne fera apparaître que les fichiers .WAV. Une fois que l'utilisateur a fait son choix, le nom du fichier doit être affecté à la propriété `Filename`.

Chapitre 15

Quiz

1. Les modèles de feuilles vous permettent d'ajouter des feuilles standards à vos applications et d'accélérer le développement de vos programmes.
2. Vous pouvez ajouter les modèles de feuilles à partir de l'assistant Création d'applications, de l'option de menu Projet, Ajouter une feuille, ou en cliquant avec le bouton droit dans la fenêtre Projet et en sélectionnant Ajouter, Feuille dans le menu contextuel.
3. Ajoutez simplement la méthode `Show` à la feuille boîte de dialogue A propos de lorsque l'utilisateur sélectionne Aide, A propos de dans le menu de votre application.
4. Faux. La boîte de dialogue A propos de contient déjà le code nécessaire à l'affichage des Infos système.
5. Un écran d'accueil ne doit rester à l'écran qu'un temps limité au démarrage du programme.
6. La boîte de dialogue Astuce du jour reste à l'écran tant que l'utilisateur le désire, alors que l'écran d'accueil n'apparaît que brièvement à titre d'introduction. De plus, l'écran d'Astuce du jour continue à afficher des astuces supplémentaires quand l'utilisateur clique sur le bouton Astuce suivante. L'utilisateur peut désactiver un écran Astuce du jour pour qu'il ne s'affiche plus au démarrage de l'application, alors que l'écran d'accueil s'affiche systématiquement au démarrage.
7. L'entrée `SaveSetting`, utiliser conjointement à la boîte de dialogue Astuce du jour spécifie si l'utilisateur souhaite la voir ou pas.

8. *ODBC* (*Open Database Connectivity*) signifie "connexion ouverte aux bases de données". Elle offre une méthode standard d'accès aux bases de données de différents systèmes informatiques.
9. Vous devez faire de la boîte de dialogue Astuce du jour ou de l'écran d'accueil la feuille de démarrage.
10. Le fichier des astuces est un fichier texte créé à l'aide d'un éditeur tel le Bloc-notes de Windows. Il contient une ligne par astuce. Vous devez enregistrer ce fichier dans le même dossier que l'application qui l'utilise.

Exercices

1. Le texte du chapitre décrit comment faire cette modification.
2. La réponse de ce projet est déjà contenue dans l'indice donné par la question. Le plus compliqué est de créer l'ensemble des astuces à l'aide du Bloc-notes.

Chapitre 16

Quiz

1. Lorsque vous liez un objet OLE, ce dernier reste avec son application parente. Lorsque vous incorporez un objet, votre application en obtient une copie : si l'objet incorporé est modifié dans son application parente, la modification n'apparaîtra pas dans votre application.
2. L'incorporation occupe le plus d'espace disque car une copie de l'objet doit se trouver dans votre application.
3. Faux. Vous devez écrire le code d'enregistrement et de chargement des modifications effectuées sur l'objet OLE.
4. La méthode `SaveToFile` enregistre un objet sur le disque.
5. La méthode `ReadFromFile` charge des objets depuis le disque.
6. L'instruction `If TypeOf` et la fonction `TypeOf()` testent la classe des objets.
7. Faux. Les objets système sont déjà globaux ; vous n'avez pas à les transmettre entre procédures.
8. Les fonctions, les constantes nommées, les fonctions internes, les procédures et les classes apparaissent souvent dans la liste des Membres.

9. Visual Basic regroupe tous les membres et les classes suivant leur usage et non pas alphabétiquement dans la liste des Membres.
10. Faux. Une des caractéristiques les plus gratifiantes de l'Explorateur d'objets est sa capacité à trouver les objets de vos applications.

Exercices

1. La clause `With` n'épargne aucun effort de programmation lorsqu'il n'y a que deux propriétés à configurer.
2. Suivez les instructions de cette leçon pour ajouter un objet WordPad à une application (mais ajoutez à la place l'objet Paintbrush). Vous devez utiliser le code d'enregistrement et de chargement des données OLE illustré dans la leçon pour pouvoir enregistrer vos dessins. Si cet exercice ne demande que peu en matière de réponse, vous pourrez acquérir, en le réalisant, une bonne expérience de l'utilisation d'objets OLE dans d'autres applications.

Chapitre 17

Quiz

1. L'*Automatisation* est le processus d'emprunt des fonctionnalités d'une autre application pour créer l'objet de donnée de cette dernière.
2. Une autre instance de Word sera démarrée et beaucoup de ressources du système seront utilisées par les deux processus redondants.
3. Vous ne pouvez pas enregistrer des applications dans des variables ; vous devez donc créer une référence à une application à l'aide de la commande `Set`.
4. Si vous déroutez une erreur à l'aide de l'instruction `On Error`, vous pouvez déterminer l'erreur survenue en testant la valeur de `Err.Number`.
5. Vous pouvez créer des contrôles ActiveX en sous-classant des contrôles uniques, des agrégats de contrôles ou en les créant de toutes pièces.
6. Le contrôle ActiveX le plus simple que vous pouvez créer est celui qui sous-classe un contrôle unique.
7. Vrai.

8. Les blocs d'énumérations sont utilisés pour définir des listes de contantes énumérées.
9. Visual Basic utilise l'extension .OCX pour les contrôles ActiveX que vous créez.
10. Les contrôles ActiveX exigent au moins une procédure `Get` et une procédure `Let` pour que vous puissiez assigner et lire les valeurs de propriétés.

Exercices

1. Aucune réponse n'est nécessaire.
2. Vous devez d'abord déclarer une variable pour contenir le texte d'origine. Vous pourrez ensuite y enregistrer le texte avant de le convertir en majuscules ou en minuscules. Vous pourriez créer une variable publique qui conserve sa valeur quand la procédure de conversion de la casse se termine.

   Etant donné que les variables locales sont préférables, le choix d'une variable publique laisse beaucoup à désirer. Il existe plusieurs méthodes pour sauvegarder la valeur d'origine de la zone de texte, mais la plus simple est sans doute de placer une zone de texte masquée dans la feuille (laissez la propriété `Enabled` à `True`, mais mettez la propriété `Visible` à `False`). Dès que l'utilisateur saisit du texte dans le contrôle ActiveX, la propriété `Text` de ce dernier doit être sauvegardée dans propriété `Text` de la zone masquée. Quand l'utilisateur clique sur le bouton de commande Comme saisi, la propriété `Text` du contrôle ActiveX doit recevoir la valeur de la propriété `Text` du contrôle masqué. *Indice* : Assurez-vous que vous mettez bien à jour la propriété `Text` de la zone de texte masquée à chaque fois que l'utilisateur tape une nouvelle valeur dans le contrôle ActiveX (comme le fait la procédure événementielle `Change`).

Chapitre 18

Quiz

1. Le Gestionnaire de données vous permet d'analyser des bases de données.
2. Une table est un fichier de données qui se trouve dans une base de données.
3. Faux. Le nombre de colonnes n'augmente pas sauf si vous augmentez le nombre de champs de la base de données.
4. Vrai. Une table est une forme de jeu d'enregistrements recordset.

5. Un *contrôle lié* est lié à un contrôle de base de données, comme le contrôle Data, qui affiche les enregistrements quand l'utilisateur parcourt la base de données.

6. Un *recordset* est un regroupement d'enregistrements, comprenant les (mais ne se limitant pas aux) enregistrements d'une table. Un *dynaset* est un regroupement d'enregistrements qui diffèrent du classement par défaut, par exemple en répondant à un critère particulier. Le dynaset change si la base est modifiée. Un *snapshot* est un instantané de dynaset, figé, qui contient certains enregistrements de la base de données tels qu'ils se présentaient à l'instant où il a été créé.

7. ADO est plus rapide, plus puissant et plus souple que le contrôle de données.

8. `EOF` détermine la fin d'une table et `BOF`, son début.

9. Une vue principale est un enregistrement et une vue secondaire est un ensemble d'enregistrements qui vont avec l'enregistrement principal. Il existe une relation un-à-plusieurs dans l'affichage Principal/secondaire ; par exemple, un vendeur peut avoir vendu plusieurs produits à votre entreprise.

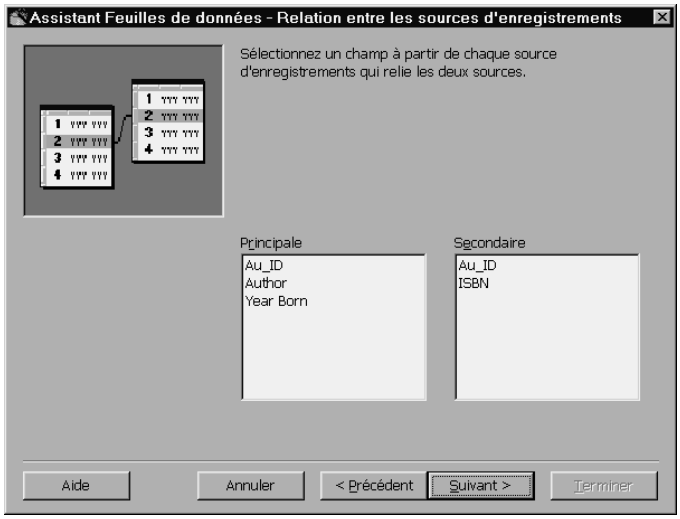
10. L'assistant Création d'applications de Visual Basic génère directement les feuilles à partir de la structure des tables de votre base de données.

Exercices

1. N'ajoutez pas de contrôle Data à la feuille. Il vous suffit d'ajouter des contrôles de zones de textes et de les lier au contrôle Data déjà présent. Configurez le contrôle Data dans la propriété `DataSource` de chaque contrôle TextBox et définissez dans la propriété `DataField` la colonne (champ) correspondante de la table.

2. Pour former une relation un-à-plusieurs, vous devez sélectionner deux tables qui ont au moins un champ commun. Quand vous sélectionnez le type de feuille Principal/secondaire, la fenêtre suivante vous donne la possibilité de sélectionner la table à utiliser comme source d'enregistrements principale. Sélectionnez la table Authors dans la liste et envoyez le champ Author dans la liste de droite pour que seul le nom de l'auteur apparaisse dans l'affichage principal de l'enregistrement. Une fois que vous avez cliqué sur Suivant, la fenêtre de source des enregistrements secondaires s'affiche. Vous pouvez y sélectionner la table Title Author. Envoyez le champ ISBN dans la liste de droite et cliquez sur Suivant pour afficher la fenêtre illustrée à la Figure A.1. Mettrez en surbrillance le champ `Au_ID` (le seul champ commun) pour connecter les enregistrements. Lorsque vous exécutez l'application, le nom de l'auteur s'affiche en haut de la feuille, et les codes ISBN de chaque ouvrage écrits par l'auteur suivent dans la partie inférieure.

Figure A.1
Vous devez indiquer à l'assistant comment lier les deux ensembles de champs.



Chapitre 19

Quiz

1. Le navigateur Web généré dans l'application affichera la page Web à l'URL que vous avez fourni lorsque l'application se connecte à l'Internet.
2. Faux. Votre application contiendra un navigateur Web incorporé utilisable et ne dépend pas de l'installation d'un navigateur sur la machine de l'utilisateur.
3. Vrai. Vous devez disposer de Internet Explorer 4 sur votre machine pour utiliser les fonctions complètes de programmation Web de Visual Basic. Ce navigateur est fourni avec Visual Basic (si vous ne l'avez pas déjà). Il n'est pas nécessaire d'en faire votre navigateur par défaut, mais c'est celui dont a besoin Visual Basic pour le développement.
4. L'encapsulation, dans sa forme la plus élémentaire, désigne l'empaquetage du code et des propriétés pour qu'un objet puisse transporter ses comportements et descriptions propres.
5. Certains contrôles de Visual Basic 6 fournissent un accès direct au service en ligne Microsoft Network. Naturellement, si l'utilisateur n'a pas l'autorisation d'accès à Microsoft Network, les contrôles ne lui permettront pas de se connecter.

6. Un intranet est une connexion interne de type Internet entre ordinateurs en réseau.
7. Un document ActiveX est une page Web qui contient une application Visual Basic entièrement fonctionnelle.
8. Java est un langage de programmation de type C++ qui permet d'écrire de petits programmes, désignés sous le terme d'applets, qui voyagent avec la page Web et s'exécutent sur la machine de l'utilisateur final quand il affiche la page Web qui les contient.
9. Le langage VBScript fonctionne avec HTML pour charger les documents ActiveX.
10. Utilisez l'assistant Migration de document ActiveX pour convertir des applications existantes en documents ActiveX.

Exercices

1. Après avoir converti une application en un document ActiveX, vous pouvez ajouter ces documents à l'application Classeur Office, qui n'est rien d'autre qu'un conteneur de contrôles ActiveX. Tant que vous ne savez pas comment compiler un fichier EXE (il faut attendre le Chapitre 21), vous devrez faire vos tests avec le raccourci Office en laissant Visual Basic en exécution. Une fois la conversion en document ActiveX effectuée, démarrez le programme Classeur Office, affichez le menu Section, et cliquez sur Ajouter pour afficher la boîte de dialogue Ajouter la sélection. Cherchez votre document ActiveX et double-cliquez dessus ; il s'affichera dans la collection des outils. Dans un projet Office, vous pourrez donc intégrer des applications non Office avec les documents Office.
2. Avec les feuilles multiples, l'assistant Migration devra créer plusieurs documents ActiveX. Vous devez tester chacun séparément en lançant votre navigateur et en sélectionnant Fichier, Ouvrir pour chaque document ActiveX créé à partir des feuilles du projet.

Chapitre 20

Quiz

1. Vous devez utiliser des fichiers RTF (*Rich Text Format*) pour le texte de l'aide et un fichier de projet HPJ, sauf si vous utilisez l'aide HTML, auquel cas le fichier d'aide aura pour extension CHM.
2. L'utilisateur peut passer d'un sujet d'aide au suivant sans revenir à un index.

3. L'étiquette de note de bas de page `K` est utilisée pour connecter l'aide aux sujets soulignés.
4. Le fichier de projet d'aide identifie le fichier de contenu, les ID de contexte des pages d'aide et le texte de la barre de titre du moteur d'aide. Le compilateur d'aide utilise le fichier de projet durant la compilation du fichier d'aide.
5. Utilisez la boîte de dialogue Propriétés de projet pour attacher l'aide à la touche F1.
6. Utilisez les valeurs d'ID de contexte pour affecter des sujets d'aide particuliers aux objets d'aide contextuelle.
7. Faux. L'aide contextuelle utilise des ID de contexte numériques.
8. Les info-bulles surgissent quand l'utilisateur laisse le curseur de la souris sur un élément. L'aide "Qu'est-ce que c'est ?" ne s'affiche que si elle est appelée à partir du menu d'aide ou si l'utilisateur clique sur le bouton "Qu'est-ce que c'est ?" de la barre d'outils.
9. Paramétrez la propriété `WhatsThisButton` de la feuille à `True` pour activer le bouton "Qu'est-ce que c'est ?" dans la barre de titre de la feuille.
10. Vrai.

Exercice

Développez l'exemple de cette leçon. L'ajout des autres sujets d'aide sera simple. Suivez ces règles :

1. Assurez-vous que la propriété `ToolTips` de chaque objet a une valeur.
2. Créez un message d'aide contextuelle pour chaque objet de la feuille.
3. Assignez des valeurs d'ID de contexte à chaque sujet d'aide, dans le fichier projet.
4. Compilez le fichier d'aide.

Chapitre 21

Quiz

1. Une erreur de syntaxe est une erreur d'orthographe ou une mauvaise utilisation du langage.

2. L'ordinateur détecte les erreurs de syntaxe, mais la correction des erreurs de logique est à la charge des personnes.
3. Vrai.
4. Faux. S'il reste une erreur de syntaxe dans l'application, Visual Basic empêche son exécution.
5. Le débogueur Visual Basic vous permet d'arrêter un programme à tout endroit en plaçant un point d'arrêt dans le code. Vous pouvez alors poser le curseur de la souris sur une variable pour voir sa valeur à l'exécution ou placer la variable dans la fenêtre Espions.
6. En exécutant un programme ligne par ligne, vous pouvez analyser les variables et contrôler les valeurs à votre rythme, et vérifier votre logique et le déroulement du programme.
7. Vrai. Vous pouvez placer des instructions d'assignation dans la fenêtre Exécution.
8. Un programme Visual Basic compilé s'exécute bien plus rapidement que dans l'environnement Visual Basic.
9. L'assistant Empaquetage et déploiement crée les routines d'installation de vos applications.
10. L'assistant Empaquetage et déploiement peut créer un ensemble de fichiers pour une installation sur plusieurs disques.

Exercices

1. L'instruction contient une erreur de syntaxe, car `Therefore` est utilisé à la place de `Then` dans l'instruction `If` de la première ligne.
2. Vrai. La phrase a non seulement une erreur de syntaxe, mais elle présente aussi une erreur de logique. En effet, la phrase n'a qu'une erreur et pas deux. Sa logique est donc incorrecte !

Chapitre 22

Quiz

1. Faux. Tous les éléments d'un tableau, quelle que soit sa dimension, doivent être du même type de données.

2. Le second indice (`9`) détermine habituellement le nombre de ligne d'un tableau multidimensionnel.
3. a. 4

   b. 43

   c. 12
4. Vrai.
5. Faux. `Array()` ne fonctionne qu'avec les tableaux à une dimension.
6. Le contrôle grille affiche les données de table efficacement.
7. Les lignes et les colonnes fixes servent de cellules de titre de la grille.
8. Utilisez une boucle imbriquée pour assigner les éléments de la table à la grille.
9. Faux. Vous devez assigner la propriété `CellPicture` à l'exécution en utilisant la fonction interne `LoadPicture()` ou un contrôle image.
10. `FormatString` est plus simple d'utilisation et est plus efficace que des instructions d'assignation pour configurer les titres des en-têtes de la grille.

Exercices

1. 28 éléments sont réservés.
2. 30 éléments sont réservés (n'oubliez pas l'indice zéro).
3. Le meilleur endroit pour enregistrer les valeurs de la grille dans la table est sans doute la procédure `cmdExit_Click()`, telle qu'illustrée ci-dessous :

```
Private Sub cmdExit_Click()
  ' Enregistre les valeurs et termine l'application
  Dim curData(19, 7) As Currency
  ' Remplit la table à partir des données de grille
  For Row = 1 To 19
    For Column = 1 To 7
      grdSales.Row = Row
      grdSales.Col = Column
      curData(Row, Column) = grdSales(Row, Column)
    Next Column
  Next Row
  ' Quitte le programme
  End
End Sub
```

Chapitre 23

Quiz

1. API signifie *Application Programming Interface*, soit Interface de programmation d'application.
2. Si Visual Basic comprend de nombreuses fonctions, il ne sait pas tout faire. Par exemple, il ne comporte pas de fonction permettant de réamorcer le système. Les routines de l'API Windows fournissent des fonctions système auxquelles vous pouvez accéder depuis votre application Visual Basic.
3. Les DLL sont liées dynamiquement à votre application au moment de l'exécution, et pas de la compilation. Ce qui signifie que les ressources de la DLL ne sont pas mises de côté par le système d'exploitation pendant toute la durée d'exécution de votre programme.
4. Quand Windows 95 est passé dans un environnement 32 bits, les noms de fichiers des DLL standards ont été modifiés. Pour distinguer les nouveaux fichiers des anciens, Microsoft a ajouté 32 dans leur nom (par exemple, GDI32.DLL).
5. La Visionneuse d'API est un outil que vous pouvez ajouter à votre environnement Visual Basic (à l'aide du menu Compléments). Il permet de sélectionner les routines API dans des listes et d'en consulter la déclaration. Vous pouvez alors les copier et les coller dans vos propres applications.
6. L'instruction `Declare` déclare les routines de l'API Windows.
7. Faux. Il n'y a pratiquement rien de normalisé dans les routines de l'API. Même parmi des familles de routines semblables, les listes d'arguments peuvent différer considérablement, comme les types de données renvoyés dans le cas des fonctions.
8. L'instruction `Declare` informe Visual Basic de l'endroit où se trouve la routine externe de l'API Windows et de sa liste d'arguments. Visual Basic ne reconnaît pas ces routines externes car il contient son ensemble propre de fonctions internes, qui sont différentes des procédures Windows. L'instruction `Declare` permet à Visual Basic de localiser et de se connecter correctement à la routine API que vous avez besoin d'appeler depuis votre application.
9. Utilisez le qualifiant `Private` quand vous déclarez des routines de l'API Windows depuis un module de feuille.
10. Une procédure d'enrobage est du code Visual Basic placé autour d'un appel à l'API Windows. Vous pouvez placer ces procédures dans un module de code réutilisable quand une application Visual Basic a besoin des routines API. Au lieu de lancer la

Visionneuse d'API et de rechercher les types de données et leurs exigences, vous n'avez plus qu'à appeler la procédure Visual Basic qui enrobe la routine API. Le débogage est simplifié et vous achevez plus rapidement la création d'applications qui demandent des routines de l'API Windows.

Exercice

La fonction `GetSystemTime()` est contenue dans le fichier KERNEL32.DLL. Vous pouvez le savoir en sélectionnant la fonction dans la liste de la Visionneuse d'API et en consultant l'argument du nom de fichier qui contient la fonction.

B

Précédence des opérateurs

Le Tableau B.1 liste l'ordre de préférence des opérateurs. Il regroupe les opérateurs par type d'opération.

Tableau B.1 : Ordre des opérateurs dans Visual Basic

Arithmétiques	*De comparaison*	*Logiques*
Exponentielle (`^`)	Egalité (`=`)	`Not`
Négation (`-`)	Différent de (`<>`)	`And`
Multiplication et division (`*`, `/`)	Inférieur à (`<`)	`Or`
Division entière (`\`)	Supérieur à (`>`)	`Xor`
Modulo arithmétique (`Mod`)	Inférieur ou égal à (`<=`)	`Eqv`
Addition et soustraction (`+`, `-`)	Supérieur ou égal à (`>=`)	`Imp`
Concaténation de chaînes de caractères (`&`)	`Like`, `Is`	

C

Table des codes ASCII

Dec.	*Hexa*	*ASCII*	*Dec.*	*Hexa*	*ASCII*
0	00	null	31	1F	e
1	01	A	32	20	espace
2	02	B	33	21	!
3	03	C	34	22	"
4	04	D	35	23	#
5	05	E	36	24	$
6	06	F	37	25	%
7	07	G	38	26	&
8	08	H	39	27	'
9	09	I	40	28	(
10	0A	J	41	29	)
11	0B	K	42	2A	*
12	0C	L	43	2B	+

Dec.	*Hexa*	*ASCII*	*Dec.*	*Hexa*	*ASCII*
13	0D	M	44	2C	,
14	0E	N	45	2D	-
15	0F	O	46	2E	.
16	10	P	47	2F	/
17	11	Q	48	30	0
18	12	R	49	31	1
19	13	S	50	32	2
20	14	T	51	33	3
21	15	U	52	34	4
22	16	V	53	35	5
23	17	W	54	36	6
24	18	X	55	37	7
25	19	Y	56	38	8
26	1A	Z	57	39	9
27	1B	a	58	3A	:
28	1C	b	59	3B	;
29	1D	c	60	3C	<
30	1E	d	61	3D	=
62	3E	>	93	5D	]
63	3F	?	94	5E	^
64	40	@	95	5F	_
65	41	A	96	60	`
66	42	B	97	61	a

Dec.	*Hexa*	*ASCII*	*Dec.*	*Hexa*	*ASCII*
67	43	C	98	62	b
68	44	D	99	63	c
69	45	E	100	64	d
70	46	F	101	65	e
71	47	G	102	66	f
72	48	H	103	67	g
73	49	I	104	68	h
74	4A	J	105	69	i
75	4B	K	106	6A	j
76	4C	L	107	6B	k
77	4D	M	108	6C	l
78	4E	N	109	6D	m
79	4F	O	110	6E	n
80	50	P	111	6F	o
81	51	Q	112	70	p
82	52	R	113	71	q
83	53	S	114	72	r
84	54	T	115	73	s
85	55	U	116	74	t
86	56	V	117	75	u
87	57	W	118	76	v
88	58	X	119	77	w
89	59	Y	120	78	x

Dec.	*Hexa*	*ASCII*	*Dec.*	*Hexa*	*ASCII*
90	5A	Z	121	79	y
91	5B	[	122	7A	z
92	5C	\	123	7B	{
124	7C	|	155	9B	õ
125	7D	}	156	9C	ú
126	7E	~	157	9D	ù
127	7F	f	158	9E	û
128	80	Ä	159	9F	ü
129	81	Å	160	A0	†
130	82	Ç	161	A1	°
131	83	É	162	A2	¢
132	84	Ñ	163	A3	£
133	85	Ö	164	A4	§
134	86	Ü	165	A5	•
135	87	á	166	A6	¶
136	88	à	167	A7	ß
137	89	â	168	A8	®
138	8A	ä	169	A9	©
139	8B	ã	170	AA	™
140	8C	å	171	AB	´
141	8D	ç	172	AC	¨
142	8E	é	173	AD	≠
143	8F	è	174	AE	Æ

Dec.	*Hexa*	*ASCII*	*Dec.*	*Hexa*	*ASCII*
144	90	ê	175	AF	Ø
145	91	ë	176	B0	∈
146	92	í	177	B1	∈
147	93	ì	178	B2	∈
148	94	î	179	B3	≥
149	95	ï	180	B4	¥
150	96	ñ	181	B5	μ
151	97	ó	182	B6	∂
152	98	ò	183	B7	$\sum$
153	99	ô	184	B8	$\prod$
154	9A	ö	185	B9	π
186	BA	∫	217	D9	Ÿ
187	BB	ª	218	DA	⁄
188	BC	º	219	DB	∈
189	BD	Ω	220	DC	‹
190	BE	æ	221	DD	›
191	BF	ø	222	DE	fi
192	C0	¿	223	DF	fl
193	C1	¡	224	E0	‡
194	C2	¬	225	E1	·
195	C3	√	226	E2	‚
196	C4	ƒ	227	E3	„
197	C5	≈	228	E4	‰

Dec.	*Hexa*	*ASCII*	*Dec.*	*Hexa*	*ASCII*
198	C6	∆	229	E5	Â
199	C7	“	230	E6	Ê
200	C8	”	231	E7	Á
201	C9	…	232	E8	Ë
202	CA	g	233	E9	È
203	CB	À	234	EA	Í
204	CC	Ã	235	EB	Î
205	CD	Õ	236	EC	Ï
206	CE	Œ	237	ED	o
207	CF	œ	238	EE	Ó
208	D0	–	239	EF	Ô
209	D1	—	240	F0	
210	D2	“	241	F1	Ò
211	D3	”	242	F2	Ú
212	D4	‘	243	F3	Û
213	D5	’	244	F4	Ù
214	D6	÷	245	F5	ı
215	D7	◊	246	F6	ˆ
216	D8	ÿ	247	F7	˜
248	F8	¯	252	FC	¸
249	F9	˘	253	FD	
250	FA	˙	254	FE	˛
251	FB	°	255	FF	

Index

Symbols

Nombres

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Le Programmeur

Microsoft® Visual Basic® 6

Grâce à cet ouvrage, vous maîtriserez très rapidement les techniques de programmation en Visual Basic 6, des plus simples aux plus complexes. Un enseignement progressif et des exercices pratiques vous aideront à concevoir des interfaces utilisateur conviviales et à créer des programmes efficaces et faciles à maintenir.

Vous découvrirez également toutes les informations nécessaires pour utiliser les contrôles ActiveX, déboguer vos applications et y intégrer les technologies d'accès aux bases de données et à Internet, ainsi que des images et des éléments multimédias.

Niveau : Débutant / Intermédiaire

Catégorie : Programmation
Configuration : Windows / Linux

- Présentation de Visual Basic
- L'environnement et les outils Visual Basic
- Gestion des contrôles
- Contrôles, propriétés et événements
- Création de menus
- Analyse des données
- Variables et expressions
- Opérateurs et instructions de contrôle
- Support avancé du clavier et de l'écran
- Entrées utilisateur et logique conditionnelle
- Sous-routines et fonctions
- Les boîtes de dialogue
- Gestion de la souris et contrôles avancés
- Sélections multiples dans une zone de liste
- Pratique de la souris
- Gestion des feuilles
- Gestion des fichiers
- Lire et écrire des fichiers
- Gestion de l'imprimante
- Image et multimédia
- Les barres de défilement
- Les modèles de feuilles
- Visual Basic et les objets
- Contrôles ActiveX
- Ces éléments qui enjolivent les applications
- Interactions avec les données
- Contrôles ADO
- Ajout d'un accès Internet
- Fournir de l'aide
- Distribution de vos applications
- Tableaux multidimensionnels
- L'API Windows

PEARSON

Pearson Education France
47 bis, rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

ISBN : 978-2-7440-4082-5

9 782744 040825