

Le Programmeur

Microsoft®

Excel® et VBA

Développez
des macros
compatibles
avec toutes
les versions
d'Excel
(de 1997
à 2010)



Codes sources
sur www.pearson.fr

Mikaël Bidault

PEARSON

LE PROGRAMMEUR

Microsoft®

Excel® & VBA

Mikaël Bidault

PEARSON

The Pearson logo consists of the word "PEARSON" in a serif font, with a thin, curved line underneath it.

Pearson Education France a apporté le plus grand soin à la réalisation de ce livre afin de vous fournir une information complète et fiable. Cependant, Pearson Education France n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Les exemples ou les programmes présents dans cet ouvrage sont fournis pour illustrer les descriptions théoriques. Ils ne sont en aucun cas destinés à une utilisation commerciale ou professionnelle.

Pearson Education France ne pourra en aucun cas être tenu pour responsable des préjudices ou dommages de quelque nature que ce soit pouvant résulter de l'utilisation de ces exemples ou programmes.

Tous les noms de produits ou marques cités dans ce livre sont des marques déposées par leurs propriétaires respectifs.

Publié par Pearson Education France
47 bis, rue des Vinaigriers
75010 PARIS
Tél. : 01 72 74 90 00
www.pearson.fr

Mise en pages : TyPAO
Collaboration éditoriale : Hervé Guyader

ISBN : 978-2-7440-4158-7
Copyright © 2010 Pearson Education France
Tous droits réservés

Aucune représentation ou reproduction, même partielle, autre que celles prévues à l'article L. 122-5 2° et 3° a) du code de la propriété intellectuelle ne peut être faite sans l'autorisation expresse de Pearson Education France ou, le cas échéant, sans le respect des modalités prévues à l'article L. 122-10 dudit code.

Sommaire

Introduction.....	1	11. Intégrer des applications VBA dans l'interface d'Excel.....	317
Partie I – Découvrir la programmation Excel	11	Partie III – Développer des interfaces utilisateur	327
1. Notions fondamentales de la programmation orientée objet	13	12. Créer des interfaces utilisateur.....	329
2. Premières macros.....	35	13. Exploiter les propriétés des contrôles ActiveX.....	359
3. Déplacement et sélection dans une macro Excel	63	14. Maîtriser le comportement des contrôles.....	401
4. Découvrir Visual Basic Editor	85	Partie IV – Notions avancées de la programmation Excel	437
Partie II – Programmeren Visual Basic	127	15. Programmer des événements Excel	439
5. Développer dans Visual Basic Editor.....	129	16. Protéger et authentifier des projets VBA	451
6. Variables et constantes.....	173	17. Exemple complet d'application Excel	475
7. Contrôler les programmes VBA	209	Annexe	
8. Fonctions Excel et VBA	257	Mots clés pour la manipulation de fichiers et de dossiers	535
9. Manipulation des chaînes de caractères.....	275	Index	539
10. Débogage et gestion des erreurs	293		

Table des matières

Introduction	1	2. Premières macros	35
VBA 7 : 64 bits vs 32 bits	2	Créer une macro GrasItalique	36
VBA, pour quoi faire ?	3	Afficher l'onglet Développeur	37
Des programmes	5	Démarrer l'enregistrement	38
Une application hôte et des projets	5	Enregistrer les commandes de la macro	40
Un langage de programmation	6	Exécuter la macro	40
Un environnement de travail	8	Structure de la macro	42
Conventions typographiques	9	Améliorer la macro	47
Codes sources en ligne	10	Une autre méthode d'enregistrement	49
I – Découvrir la programmation Excel	11	Enregistrement	50
1. Notions fondamentales de la programmation orientée objet	13	Structure de la macro	50
Comprendre le concept d'objet	14	Écrire la macro	51
Objets et collections d'objets	14	Exécution de la macro	53
Application hôte et modèles d'objets	16	Choisir l'accessibilité des macros	53
Accéder aux objets	19	Accessibilité globale ou limitée	53
Les propriétés	21	Classeurs et modèles	54
Les méthodes	27	Le classeur de macros personnel	55
Les événements	28	Les macros complémentaires	56
Les fonctions	29	Définir le classeur de stockage lors de l'enregistrement d'une macro	60
Le modèle d'objets d'Excel	29	Accéder aux macros d'un classeur spécifique ..	60

3. Déplacement et sélection

dans une macro Excel 63

Méthodes de sélection dans une feuille

Excel 64

Clavier..... 64

Souris 66

Notion de cellule active..... 66

Références relatives et références absolues 67

Coder les déplacements effectués lors

de l'enregistrement d'une macro 69

Référence absolue aux cellules 69

Référence relative aux cellules..... 78

Référence aux cellules fonction

de leur contenu..... 80

Référence aux plages de cellules nommées..... 83

4. Découvrir Visual Basic Editor 85

Accéder à Visual Basic Editor..... 86

Les outils et les fenêtres

de Visual Basic Editor..... 88

L'Explorateur de projet..... 89

L'Explorateur d'objets 92

La fenêtre UserForm 100

La fenêtre Code 102

La fenêtre Propriétés..... 116

Les barres d'outils..... 121

Paramétrer Visual Basic Editor 124

II – Programmer en Visual Basic 127

5. Développer dans Visual Basic Editor 129

Structure des programmes Visual Basic..... 130

Les modules..... 130

Les procédures 131

Les instructions 133

Les différents types de procédures..... 136

Procédures Sub 136

Procédures Function 141

Procédures Property 143

Des projets bien structurés..... 149

Ajouter un module..... 149

Supprimer un module 151

Créer une procédure 152

Écrire l'instruction de déclaration..... 152

La boîte de dialogue Ajouter une procédure.... 153

La notion de portée 154

Écriture et mise en forme du code..... 155

Déplacer une procédure 161

Appel et sortie d'une procédure..... 162

Appel d'une procédure Sub..... 162

Appels de procédures Function et Property 163

Passage d'arguments..... 164

Sortie d'une procédure 166

Sortie d'un programme 167

Exécuter du code..... 168

Aide à l'écriture de code..... 169

Vérification automatique de la syntaxe 169

Complément automatique des instructions 170

Info express automatique 171

6. Variables et constantes..... 173

Déclarer une variable 174

Déclaration implicite 174

Déclaration explicite..... 175

Types de données des variables 178

Chaînes de caractères 178

Valeurs numériques 180

Valeurs booléennes..... 183

Dates 184

Type Variant..... 185

Variables de matrice..... 185

Variables objet..... 190

Types de données personnalisés..... 195

<i>Constantes</i>	197	Recommandations pour l'écriture de fonctions	
<i>Validation et conversion des types de données</i>	198	Excel	263
Portée et durée de vie des variables	201	<i>Les limites de la cellule</i>	263
<i>Portée de niveau procédure</i>	201	Principales fonctions VBA.....	264
<i>Portée de niveau module privée</i>	201	9. Manipulation des chaînes de caractères	275
<i>Portée de niveau module publique</i>	202	Modifier des chaînes de caractères	276
<i>Variables statiques</i>	203	<i>Concaténer des chaînes</i>	276
Traitement interapplications à l'aide		<i>Insérer des caractères non accessibles au clavier</i>	
de variables objet	203		278
7. Contrôler les programmes VBA	209	<i>Répéter une série de caractères</i>	280
Répéter une série d'instructions : les boucles	210	<i>Supprimer les espaces superflus d'une chaîne</i>	280
<i>La boucle While...Wend</i>	210	<i>Extraire une partie d'une chaîne</i>	281
<i>La boucle Do...Loop</i>	215	<i>Effectuer des remplacements</i>	
<i>La boucle For...Next</i>	218	<i>au sein d'une chaîne</i>	282
<i>La boucle For Each...Next</i>	224	<i>Modifier la casse des chaînes de caractères</i>	283
Utiliser des instructions conditionnelles	228	Comparer des chaînes de caractères	283
<i>La structure de contrôle If...Then...Else</i>	228	Rechercher dans les chaînes de caractères.....	286
<i>La structure de contrôle Select Case</i>	233	<i>Rechercher une chaîne dans une chaîne</i>	286
Définir l'instruction suivante avec GoTo	234	<i>Rechercher une chaîne dans une variable</i>	
Interagir avec l'utilisateur via des boîtes		<i>de matrice</i>	289
de dialogue	235	10. Débogage et gestion des erreurs.....	293
<i>La fonction InputBox</i>	235	Les étapes et les outils du débogage	294
<i>La méthode InputBox</i>	239	<i>Test du projet</i>	295
<i>La fonction MsgBox</i>	241	<i>Exécuter pas à pas</i>	297
<i>Affichage de boîtes de dialogue Excel</i>	246	<i>La fenêtre Variables locales</i>	298
Utiliser les opérateurs logiques.....	252	<i>Les points d'arrêt</i>	300
Trier des données	253	<i>Modifier l'ordre d'exécution des instructions ..</i>	301
8. Fonctions Excel et VBA	257	<i>La fenêtre Exécution</i>	301
Utiliser les fonctions Excel dans VBA.....	258	<i>Les espions</i>	302
Créer des fonctions Excel personnalisées	258	<i>La pile des appels</i>	304
Intégrer une fonction via l'Explorateur		Exemple de débogage	305
d'objets.....	260	<i>Recherche du bogue</i>	307
<i>Insérer une fonction VBA dans votre code</i>	260	<i>Résolution du bogue</i>	308
<i>Insérer une fonction Excel dans votre code</i>	261	Gestion des erreurs et des exceptions	312
		Exemple de gestion d'erreur	313

11. Intégrer des applications VBA dans l'interface

d'Excel.....	317
Affecter une macro à un raccourci clavier	318
Personnaliser le ruban et la barre d'outils	
Accès rapide.....	319
Personnaliser les barres d'outils	
dans les versions antérieures à Excel 2007	321
Personnaliser les menus dans les versions	
antérieures à Excel 2007	323
Affecter une macro à un objet.....	325

III – Développer des interfaces utilisateur 327

12. Créer des interfaces

utilisateur	329
Les phases de développement de feuilles	331
Créer une feuille.....	331
Les contrôles de la boîte à outils.....	333
<i>Outil Sélection</i>	334
<i>Contrôle Label</i>	334
<i>Contrôle TextBox</i>	334
<i>Contrôle ComboBox</i>	335
<i>Contrôle Frame</i>	335
<i>Contrôle ListBox</i>	336
<i>Contrôle CheckBox</i>	336
<i>Contrôle OptionButton</i>	337
<i>Contrôle ToggleButton</i>	338
<i>Contrôle CommandButton</i>	338
<i>Contrôle TabStrip</i>	338
<i>Contrôle MultiPage</i>	339
<i>Contrôle ScrollBar</i>	340
<i>Contrôle SpinButton</i>	340
Placer des contrôles sur une feuille.....	341
<i>Copier-coller des contrôles</i>	344
<i>Sélectionner plusieurs contrôles</i>	345
<i>Supprimer des contrôles</i>	345

Mise en forme des contrôles	346
<i>La grille</i>	346
<i>Aligner les contrôles</i>	347
<i>Uniformiser la taille des contrôles</i>	348
<i>Uniformiser l'espace entre les contrôles</i>	349
<i>Centrer les contrôles</i>	350
<i>Réorganiser les boutons</i>	351
<i>Grouper ou séparer des contrôles</i>	351
Personnaliser la boîte à outils	352
<i>Ajouter/supprimer un contrôle</i>	353
<i>Ajouter/supprimer une page</i>	355
Afficher/masquer une feuille.....	357

13. Exploiter les propriétés

des contrôles ActiveX	359
Propriété Name	361
Apparence	362
Alignment.....	362
BackColor	363
BackStyle	363
BorderColor	363
BorderStyle	364
Caption	364
ControlTipText	365
ForeColor	365
SpecialEffect	366
Style	366
Value.....	367
Visible	369
Comportement.....	372
AutoSize.....	372
AutoTab.....	373
AutoWordSelect	374
Cancel.....	374
Default.....	375
Enabled.....	375
EnterKeyBehavior	377

Police..... 399

Font..... 399

14. Maîtriser le comportement des contrôles 401

Créer des procédures événementielles	402
<i>Créer une procédure</i>	402
<i>Les événements</i>	408
Exemples d'exploitation des contrôles	415
<i>Contrôle Label</i>	415
<i>Contrôle TextBox</i>	418
<i>Contrôle ComboBox</i>	421
<i>Contrôle ListBox</i>	426
<i>Contrôles CheckBox et OptionButton</i>	429
<i>Contrôle ScrollBar</i>	430
<i>Contrôle SpinButton</i>	432
<i>Exploiter les informations d'une feuille VBA</i> ...	435

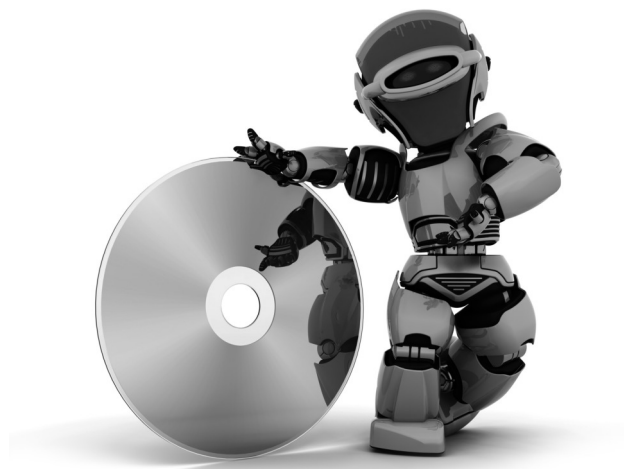
Excel 437

15. Programmer des événements Excel.....	439
L'objet Application	440
<i>Déclaration et instanciation de l'objet</i>	
Application.....	440
<i>Création de procédures événementielles</i>	
<i>de niveau application</i>	441
<i>Propriétés de l'objet Application</i>	443
<i>Méthodes de l'objet Application</i>	445
L'objet ThisWorkbook	445
L'objet Worksheet	448

16. Protéger et authentifier des projets VBA.. 451

Les virus macro.....	452
Se protéger des virus macros	452
<i>Définir un niveau de sécurité dans Excel 2010 et Excel 2007</i>	452

<i>Définir un niveau de sécurité</i> <i>avec Excel 2000, XP et 2003</i>	455	Créer un modèle Excel.....	485
<i>Les signatures numériques</i>	456	Définir et créer des interfaces	486
<i>Effectuer des sauvegardes des macros</i>	457	<i>Feuille fmContratAuteur</i>	487
Protéger l'accès aux macros.....	459	<i>Feuille fmContratConditions</i>	500
<i>Verrouiller un projet</i>	459	<i>Feuille fmContratDates</i>	514
<i>Limiter les droits d'exécution d'une macro</i>	460	<i>Feuille fmContratImpression</i>	520
Authentifier ses macros.....	472	<i>Feuille fmContratFin</i>	527
<i>Obtenir une authentification</i>	472	Écrire des procédures d'édition de documents	528
<i>Authentifier une macro</i>	473	<i>Édition des feuilles de paie</i>	529
17. Exemple complet d'application Excel	475	<i>Mise à jour du Tableau Word</i>	531
Présenter un projet d'application Excel	476	Annexe. Mots clés pour la manipulation de fichiers et de dossiers	535
<i>Avant de commencer</i>	477	Index	539
<i>Identification des informations à recueillir</i>	477		
<i>Définition de la structure du programme</i>	479		



Introduction

Visual Basic pour Applications, VBA, est la solution de programmation proposée avec les applications de la suite Office. La connaissance de VBA permet à l'utilisateur d'Excel de tirer pleinement profit du tableur de Microsoft en en développant les capacités et les fonctionnalités pour ses besoins spécifiques. Maîtriser Visual Basic pour Applications, c'est à coup sûr améliorer grandement sa productivité et celle de son entreprise.

L'intégration dans Excel de Visual Basic pour Applications, un *environnement de développement intégré* complet et professionnel, remonte à sa version 97. Depuis, Microsoft a confirmé sa volonté de faire de VBA un élément à part entière des applications Office et l'a progressivement proposé avec l'ensemble des applications de sa suite bureautique. Visual Basic pour Applications constitue aujourd'hui un langage et un environnement stables et pérennes.

Bien que certains développeurs demandaient le remplacement de VBA par VB.net, Microsoft confirme, avec Office 2010, sa volonté de maintenir VBA comme solution de programmation des applications Office, puisque la nouvelle suite Office intègre la version 7 de Visual Basic (les versions XP, 2003 et 2007 d'Office intégraient Visual Basic 6.3). Également confirmé dans cette version, le nouvel environnement apparu avec Office 2007 et le remplacement des menus "classiques" des versions précédentes par un système de "ruban" et d'"onglets".

Cet ouvrage traite de la programmation VBA d'Excel 2010, mais, sauf exception signalée, les explications et les exemples proposés sont aussi valides pour les versions 97, 2000, XP, 2003 et 2007 d'Excel. En effet, d'une version à l'autre, il n'y a pas eu de révolution. Le modèle d'objets s'est affiné et les nouvelles fonctions d'Excel, apparues au cours des différentes versions du logiciel, peuvent également être manipulées *via* la programmation VBA. Cependant, le langage, la gestion des programmes, l'environnement et les outils au

service du développeur – bref, tout ce que vous devez savoir pour programmer Excel et que cet ouvrage se propose de vous apprendre – restent inchangés d’une version à l’autre.

Donc, que vous utilisiez encore Excel 2003 ou que vous soyez passé à la version 2010, sachez que vous pourrez appliquer les connaissances que vous aurez acquises lors de la lecture de ce livre quand vous migrerez d’une version à l’autre d’Excel. Mieux, les programmes développés pour Excel 97 fonctionnent avec toutes les versions ultérieures du tableur et, sauf exception, les programmes développés dans Excel 2010 devraient fonctionner avec les versions antérieures.

Dans cet ouvrage, vous découvrirez les différentes méthodes de création de projets VBA pour Excel, Visual Basic (le langage de programmation proprement dit) et les outils de développement et de gestion intégrés de Visual Basic pour Applications. Votre initiation à la programmation VBA se fera au moyen d’exemples de programmes détaillés et commentés.



Vous rencontrerez le terme projet tout au long de cet ouvrage. C’est ainsi que l’on nomme un ensemble de programmes développés avec Visual Basic pour Applications.

VBA 7 : 64 bits vs 32 bits

Pour la première fois de son histoire, Office est proposée en deux versions : 32 bits et 64 bits. Ces deux versions sont pour ainsi dire indifférenciables : seule la gestion de la mémoire varie d’une version à l’autre, autorisant la manipulation de fichiers nettement plus volumineux avec la version 64 bits. Cependant, Microsoft recommande l’installation de la version 32 bits, y compris sur un système d’exploitation 64 bits, notamment pour des raisons de compatibilité des compléments (comme les macros VBA) avec les versions précédentes. C’est d’ailleurs la version 32 bits qui est installée par défaut, et les utilisateurs souhaitant installer la version 64 bits doivent parcourir le CD afin d’exécuter le fichier d’installation correspondant.

Conséquence pour le développement de macros Excel : la version 7 de VBA est maintenant une version 64 bits, qui intègre le support d’un nouveau type de données permettant la manipulation des pointeurs (les pointeurs permettent la manipulation des API Windows). Cet ouvrage n’abordant pas la manipulation des API – notions réservées aux programmeurs chevronnés – cette nouveauté n’a aucune incidence sur la validité de ce que vous explique ce livre. Les concepts et techniques de programmation que vous apprendrez ici sont donc compatibles avec la version 64 bits d’Office comme avec les versions 32 bits d’Office, sans qu’il soit nécessaire d’adapter le code.

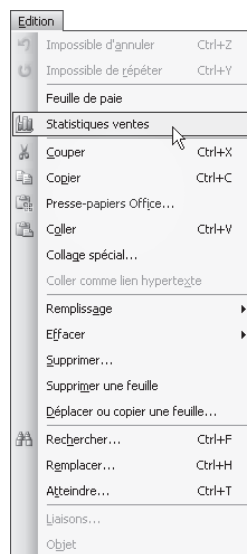
VBA, pour quoi faire ?

Excel offre des possibilités très étendues. Pourtant, quelle que soit la puissance des fonctions d'Excel, elles ne peuvent répondre à toutes les situations. La programmation VBA est la solution de personnalisation offerte par Excel, afin d'ajouter des caractéristiques, des fonctions et des commandes qui répondent précisément à vos besoins.

La programmation VBA peut être définie comme la *personnalisation d'un logiciel afin de s'assurer gain de temps, qualité des documents et simplification des tâches complexes ou fastidieuses*. Voici quelques exemples de ce que permettent les programmes VBA :

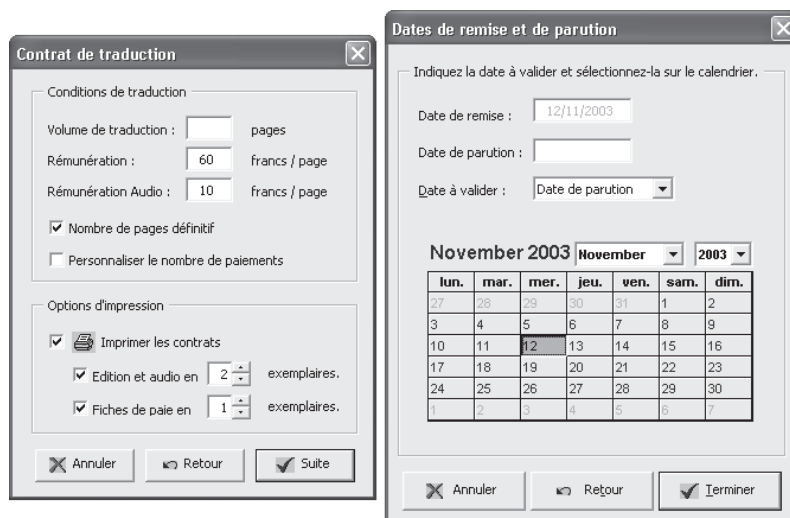
- **Combiner un nombre indéterminé de commandes.** Nous sommes souvent amenés à répéter ou à associer certaines commandes plutôt que d'autres et à ignorer certaines fonctionnalités en fonction de l'usage personnel que nous avons d'un logiciel. VBA permet d'associer un nombre illimité de commandes à une seule. Vous pouvez ainsi ouvrir simultanément plusieurs documents Excel stockés dans des dossiers ou sur des serveurs différents, y insérer des données spécifiques et leur appliquer des mises en forme adaptées, en exécutant une seule commande créée en VBA.
- **Ajouter de nouvelles commandes et de nouvelles fonctions à Excel.** VBA permet de créer de nouvelles commandes et d'ajouter des fonctions au tableur – par exemple une fonction personnalisée qui permet de calculer les taxes à retenir sur un salaire (ou, mieux, les primes à y ajouter), etc. Vous pouvez, en outre, attacher vos programmes VBA à des raccourcis clavier, à des icônes et à des commandes de menu afin d'en améliorer l'accessibilité.

Figure I.1
VBA permet de personnaliser
l'interface des applications
Office en y ajoutant icônes et
commandes de menus.



- **Automatiser des actions répétitives.** Nous sommes parfois amenés à répéter certaines opérations plusieurs fois sur un même document ou à réitérer des traitements spécifiques. Un programme VBA peut, par exemple, mettre en forme des cellules dans un classeur Excel, effectuer des séries de calculs, etc.
- **Modifier et améliorer les commandes d'une application.** Les commandes Excel ne sont pas toujours adaptées à nos besoins ou présentent parfois des limitations gênantes. Un programme VBA peut modifier, brider ou compléter les commandes d'une application. Vous pouvez ainsi intégrer dans un tableau le nom de l'utilisateur, le nombre de pages imprimées et l'imprimante utilisée chaque fois qu'une impression est lancée à partir d'Excel.
- **Faire interagir les différentes applications Office.** Un programme VBA peut exploiter des données issues de fichiers générés par d'autres programmes et interagir avec ceux-ci de façon transparente pour l'utilisateur. Vous pouvez ainsi créer une commande qui envoie automatiquement le classeur Excel ouvert en fichier joint dans un mail Outlook à des destinataires définis ou qui génère un rapport Word à partir de données Excel et l'imprime.
- **Créer des interfaces personnalisées.** Les programmes VBA peuvent ramener des tâches complexes à la simple information de champs dans des boîtes de dialogue personnalisées pour l'utilisateur final, simplifiant ainsi considérablement le travail de celui-ci, tout en vous assurant qu'aucun oubli ou fausse manipulation n'aura lieu.

Figure I.2
Visual Basic pour Applications vous permet de développer des interfaces utilisateur évoluées.



Visual Basic pour Applications permet le développement de solutions adaptées à vos besoins. Les outils que vous apprendrez à manier vous permettront de développer des programmes simples, sans écrire la moindre ligne de code, comme des programmes complets intégrant une interface utilisateur adaptée.

La fonction d'un programme VBA peut être d'automatiser une tâche répétitive. Mais vous pouvez aussi créer très vite un petit programme VBA pour faire face à une nécessité immédiate ; par exemple, afin de généraliser un traitement exceptionnel à l'ensemble d'un document.

Des programmes

Les projets VBA sont des programmes ou *macros* écrits dans le langage Visual Basic. Si vous ne possédez aucune expérience préalable de programmation, ne vous inquiétez pas : cet ouvrage aborde le développement de projets VBA à travers l'enregistrement de macros. Lorsque vous l'activez, l'Enregistreur de macro mémorise chacune de vos actions. C'est votre programmeur personnel : vous utilisez simplement les commandes d'Excel et il se charge de traduire les actions exécutées en instructions Visual Basic. Il vous suffit ensuite d'exécuter la macro pour répéter l'ensemble des commandes enregistrées.



Le terme macro désigne le regroupement d'un ensemble de commandes en une seule. On parle parfois de macrocommandes pour désigner un programme qui se résume à l'exécution d'une série de commandes, sans égard pour le contexte. Des macros plus évoluées peuvent répéter des opérations en boucle, afficher des boîtes de dialogue qui autorisent une interaction avec l'utilisateur. Ces programmes se comporteront différemment en fonction des informations entrées ou de l'état du document sur lequel elles s'exécutent.

Le terme projet est plus large. Il désigne l'ensemble des éléments constituant vos programmes VBA. Il s'agit toujours de macros, mais à celles-ci peuvent s'ajouter des feuilles – qui constituent une interface utilisateur permettant de récolter des informations de tout type –, des modules de classe, et autres friandises que vous découvrirez tout au long de cet ouvrage.

L'enregistrement de macros constitue sans aucun doute le meilleur moyen de se familiariser avec la programmation en Visual Basic. Ainsi, sans connaître le langage – les instructions qui le composent et la façon dont elles sont structurées –, vous pouvez créer des programmes VBA et en visualiser ensuite le code.

Une application hôte et des projets

Visual Basic pour Applications est un environnement de développement calqué sur Visual Basic, un outil de développement d'applications Windows. Les structures de contrôle du

langage sont les mêmes et l'environnement proprement dit (Visual Basic Editor) est pour ainsi dire identique à celui de Visual Basic. Mais, contrairement à Visual Basic, Visual Basic pour Applications est conçu... *pour des applications*. Cela signifie que, tandis que les programmes Visual Basic sont autonomes, les programmes VBA ne peuvent être exécutés qu'à partir d'une application intégrant cet environnement de développement – Excel ou une autre application.

Lorsque vous développez un programme VBA, vous l'attachez à une application. Il s'agit de l'*application hôte* du programme. Plus précisément, vos programmes VBA sont attachés à un document (un fichier ou un modèle Word, une feuille de calcul Excel, une présentation PowerPoint...) spécifique à l'application hôte. L'ensemble des programmes VBA attachés à un document constitue un projet. Un projet regroupe des macros, mais peut également intégrer des interfaces utilisateur, des déclarations système, etc. Un projet constitue en fait la partie VBA d'un document. Si cet ouvrage ne traite que de la programmation pour Excel, sachez qu'un programme VBA peut être attaché à une autre application. Les concepts et les outils que vous découvrirez au long de cet ouvrage sont valides pour toutes les applications de la suite Office. Pour exécuter une macro VBA, vous devez avoir accès au document auquel elle est attachée. Vous pouvez choisir de rendre certaines macros disponibles à partir de n'importe quel document Excel ou en limiter l'accessibilité à un classeur Excel spécifique. La disponibilité des programmes VBA est abordée au Chapitre 2.

Un langage de programmation

Les projets VBA sont développés dans le langage de programmation Visual Basic. Vous découvrirez par la pratique la structure de ce langage et apprendrez rapidement à en discerner les composants et les relations qu'ils entretiennent. Comme nous l'avons dit précédemment, l'enregistrement de macros constitue une excellente initiation à Visual Basic. C'est sous cet angle que nous vous ferons découvrir ce langage.

Visual Basic est un langage de programmation *orienté objet*. Nous présenterons donc les concepts de la programmation orientée objet (POO). Vous apprendrez ce qu'est un objet, une propriété, une méthode ou un module de classe, etc. Vous verrez comment conjuguer ces éléments pour créer des applications Excel souples et puissantes. Visual Basic pour Applications constitue une bonne approche de la programmation pour le néophyte.

Visual Basic pour Applications intègre un grand nombre d'instructions. Cela permet de développer des macros susceptibles d'identifier très précisément l'état de l'application et des documents et reproduire l'exécution de la plupart des commandes disponibles dans l'application hôte.

Vous verrez que certaines instructions sont spécifiques à Excel. C'est, par exemple, le cas des instructions permettant d'affecter une formule à une cellule. Vous n'utiliserez probablement qu'un nombre limité de ces instructions, en fonction de votre usage personnel d'Excel

ou des besoins de votre entreprise. Par ailleurs, certaines instructions spécifiques à Excel apparaîtront presque toujours dans vos macros. C'est, par exemple, le cas de la propriété `Range` qui renvoie un objet Excel tel qu'une cellule ou une plage de cellules.

D'autres instructions sont communes à l'ensemble des applications Office. C'est le cas de celles qui permettent de régler le comportement d'une macro : réaliser des opérations en boucle, induire des réactions face à certains paramètres, afficher des boîtes de dialogue simples (voir Figures I.3 et I.4) ou développer des interfaces utilisateur évoluées (voir Figure I.1), etc. Ce sont ces instructions qui constituent véritablement ce qu'il est convenu d'appeler *le langage Visual Basic*. Vous aurez besoin d'y faire appel dès que vous voudrez créer un programme interactif, capable de se comporter différemment selon le contexte. La plupart de ces instructions ne peuvent être générées par enregistrement de macros, et doivent donc être éditées manuellement dans Visual Basic Editor.

Figure I.3

La fonction VBA MsgBox permet d'afficher une boîte de dialogue.

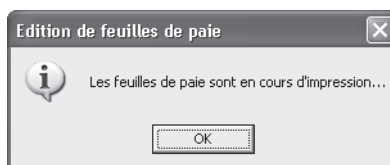


Figure I.4

Il existe une version VBA et une version Excel de la fonction InputBox.



Cet ouvrage ne se veut pas un dictionnaire du langage, mais un guide qui vous enseignera le développement de projets VBA de qualité. Vous apprendrez à enregistrer, modifier, exécuter et déboguer des macros, à créer des interfaces utilisateur ainsi qu'à gérer vos projets VBA. Vous découvrirez, à travers les nombreux exemples de projets VBA de cet ouvrage, un certain nombre d'instructions spécifiques à la *hiérarchie d'objets* d'Excel, qui vous familiariseront avec la logique de ce langage.



La hiérarchie d'objets d'une application, encore appelée modèle d'objets, est le rapport qu'entretiennent entre eux les différents objets d'une application. Ce concept ainsi que les notions spécifiques aux langages orientés objet seront développés au Chapitre 1, "Notions fondamentales de la programmation orientée objet".

En revanche, ce livre présente et illustre d'exemples commentés l'ensemble des structures de contrôle qui permettront de créer très simplement des macros évoluées. Nous vous fournirons les bases du langage Visual Basic. Elles suffisent pour créer une infinité de macros et répondre à vos besoins spécifiques.

Lorsque les principes du développement de projets VBA vous seront acquis et que vous créerez vos propres macros, il vous arrivera sûrement d'avoir besoin d'instructions que vous n'aurez pas rencontrées lors de la lecture de cet ouvrage ; vous pourrez alors utiliser l'Enregistreur de macro ou encore les rechercher dans l'aide de Visual Basic pour Applications ou dans l'Explorateur d'objets – étudié au Chapitre 4. Vous verrez que l'aide de Visual Basic pour Applications fournit une référence complète du langage, facilement accessible et consultable.

Si vous n'avez aucune expérience de programmation, peut-être ce *Visual Basic* vous apparaîtra-t-il comme un langage barbare ou inaccessible. Ne vous inquiétez pas : le développement de projets VBA ne requiert ni expérience préalable de la programmation, ni connaissance globale du langage. Contentez-vous, au cours de votre lecture, d'utiliser les fonctions nécessaires aux exercices et que nous vous détaillerons. Cet ouvrage propose un apprentissage **progressif** et **concret** : vous développerez vos premiers projets VBA dès les premiers chapitres.

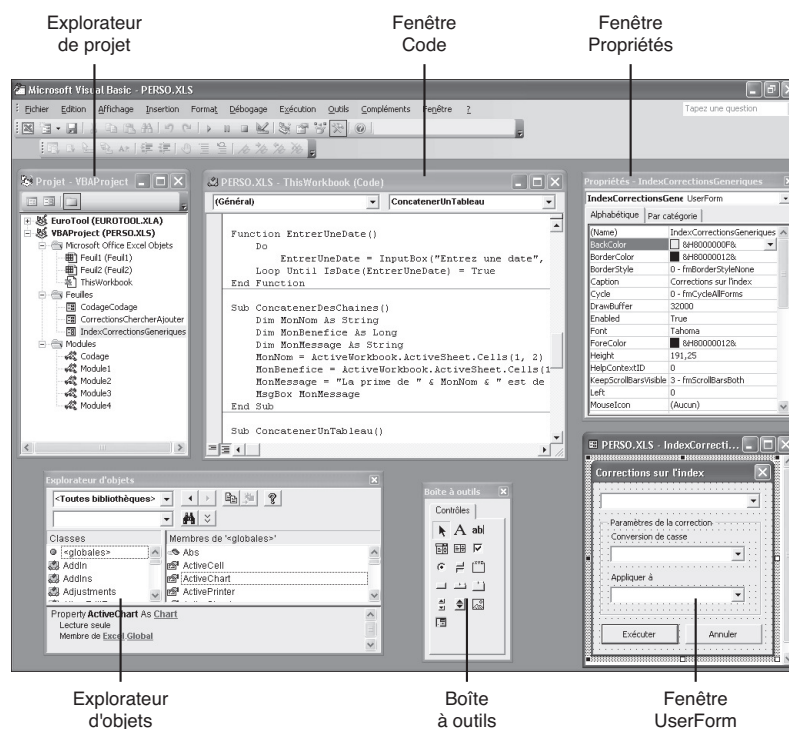
Un environnement de travail

Visual Basic pour Applications dispose d'un environnement de développement à part entière : Visual Basic Editor.

Visual Basic Editor est l'*environnement de développement intégré* des applications Office. Il permet de visualiser et de gérer les projets VBA, d'écrire, de modifier et de déboguer les macros existantes, de visualiser comment les commandes propres à une application Office sont traduites en langage Visual Basic, et inversement. C'est aussi un outil de débogage de vos projets VBA d'une grande efficacité. Visual Basic Editor propose nombre d'outils permettant de tester les macros et d'en étudier le comportement. Vous pouvez ainsi exécuter les commandes de la macro pas à pas, en suivre le déroulement, insérer des commentaires dans le texte de la macro, etc. Enfin, Visual Basic Editor intègre des outils très intuitifs, dédiés au développement d'interfaces graphiques.

Vous apprendrez dans cet ouvrage à utiliser les nombreux outils de Visual Basic Editor à toutes les phases de développement d'un projet VBA.

Figure 1.5
Visual Basic Editor
 est l'environnement
 de développement
 de Visual Basic pour
 Applications.



Conventions typographiques

Afin d'en faciliter la lecture, nous avons adopté dans cet ouvrage un certain nombre de conventions typographiques. Lorsqu'un mot apparaît pour la première fois, il est composé en *italique*. Les programmes et les mots clés du langage Visual Basic apparaissent dans une police à chasse fixe. Lorsque, dans un programme, un mot signale une information attendue dans le code, celui-ci apparaît en *italique*.

Lorsqu'une ligne de code ne peut être inscrite sur une seule ligne de l'ouvrage, cette flèche (➡) en début de ligne indique que le texte est la poursuite de ligne précédente.

Par ailleurs, vous rencontrerez au long de cet ouvrage différents types de notes, signalées dans la marge par des pictogrammes.



Ces rubriques apportent un complément d'information en rapport avec le sujet traité. Leur lecture n'est pas indispensable. Mais elles peuvent vous aider à mieux cerner le sujet.



Vous trouverez sous ces rubriques la définition de termes techniques spécifiques à la programmation VBA.



Ces rubriques vous mettent en garde contre les risques inhérents à telle ou telle commande ou manipulation.



Il est parfois nécessaire de se rafraîchir la mémoire. Lorsqu'un sujet fait appel à des connaissances acquises plusieurs chapitres auparavant, cette rubrique vous les remémore brièvement.



Sous cette rubrique, vous trouverez des trucs pour aller plus vite et travailler plus efficacement.



Nous vous faisons ici part de notre expérience, en vous prodiguant des conseils qui vous aideront à développer des projets VBA de qualité.



Ces notes prodiguent des informations spécifiques aux versions antérieures à Office 2010.

Codes sources en ligne

Vous trouverez sur le site Pearson (www.pearson.fr), à la page consacrée à cet ouvrage, les codes sources des applications VBA développées dans ce livre.



Découvrir la programmation Excel

CHAPITRE 1. Notions fondamentales de la programmation orientée objet

CHAPITRE 2. Premières macros

CHAPITRE 3. Déplacement et sélection dans une macro Excel

CHAPITRE 4. Découvrir Visual Basic Editor



1

Notions fondamentales de la programmation orientée objet

Au sommaire de ce chapitre

- Comprendre le concept d'objet
- Le modèle d'objets Excel

Visual Basic est un langage de programmation *orienté objet*. En tant que tel, il repose sur des concepts communs à tous les langages de programmation orientés objet. Avant de vous lancer dans la programmation pour Excel, il est important de vous familiariser avec ces concepts et le vocabulaire qui les décrit. Plus concrètement, ce chapitre vous fera découvrir les différents composants du langage Visual Basic en tant que langage orienté objet et comment ils s'articulent pour créer des programmes VBA puissants.

Vous ne trouverez pas dans ce chapitre de programmes VBA. Il est destiné à vous faire acquérir les bases et la terminologie sur lesquelles nous nous appuierons tout au long de cet ouvrage. Alors, patience ! Les connaissances qu'il vous apportera permettront d'appréhender vos premiers programmes dès le Chapitre 2.

Comprendre le concept d'objet

Comme pour tous les langages de programmation objet, les *objets* sont le fondement de Visual Basic. Quelle que soit la fonction d'un programme VBA, presque toutes les actions qu'il exécute s'apparentent à la modification d'objets.

Les ouvrages présentant la programmation orientée objet (POO) le font presque toujours par analogie avec les objets de la vie réelle. Nous ne dérogerons pas à cette règle. La programmation orientée objet repose en effet sur une structure qui rappelle, par de nombreux points, les objets de la vie courante et les rapports qu'ils entretiennent. L'analogie avec les objets de la vie réelle rend simples et faciles d'accès des concepts qui, abordés de façon abstraite, vous apparaîtraient probablement obscurs.

Objets et collections d'objets

Dans la vie, un objet peut être tout et n'importe quoi. Ce qui caractérise un objet, c'est son existence physique, ses propriétés spécifiques, son comportement et les actions que l'on peut exécuter sur celui-ci. Une voiture est un objet. Lorsque vous parlez de l'objet Voiture, vous pouvez faire référence à un objet abstrait ("Je vais acheter une voiture") comme à une voiture bien concrète ("Mate un peu ma belle 2CV rouge"). Les objets que vous utiliserez dans vos programmes VBA répondent à une même définition.

Dans le premier cas, vous évoquez un objet Voiture imprécis, et pourtant tout le monde comprend de quoi vous parlez. Il vous suffit de prononcer le mot "voiture" pour que chacun imagine et visualise une voiture bien spécifique, en fonction de ses goûts, de ses aspirations, de ses souvenirs, etc. Cependant, quelle que soit la voiture imaginée, en tant qu'objet Voiture, elle possède un certain nombre de *propriétés* (une carrosserie, des roues, un moteur, etc.) et autorise un certain nombre de *méthodes* (démarrer, freiner, tourner, etc.) qui permettent d'en maîtriser le comportement.

Ce sont ces *propriétés* et ces *méthodes*, communes à toutes les voitures, qui définissent l'objet Voiture. Elles sont sous-entendues, évidentes et essentielles. Il existe donc des milliers de voitures différentes, toutes reconnaissables par un certain nombre de caractéristiques communes définies dans le concept (l'objet) Voiture. En POO, cet objet abstrait est appelé la *classe* Voitures. La classe Voitures est la définition formelle des objets Voiture (leurs propriétés et leurs méthodes). Il s'agit du modèle à partir duquel vous pouvez imaginer et créer des milliers de voitures différentes. L'ensemble des véhicules appartenant à la classe Voitures (parce qu'ils possèdent les propriétés et les méthodes définies dans cette classe) est appelé la *collection d'objets* Voitures.



Une collection porte le nom pluriel des objets qu'elle rassemble.

Ainsi, la collection WorkBooks renvoie tous les objets Workbook, soit tous les classeurs ouverts, la collection Sheets, toutes les feuilles d'un objet Workbook, la propriété Worksheets, toutes les feuilles de calcul d'un objet Workbook, etc. La section "Le modèle d'objets d'Excel" située en fin de chapitre vous permettra de découvrir les objets Excel les plus importants.



Le terme Classe désigne la définition commune d'un ensemble d'objets (qu'est-ce qu'une voiture ?), tandis qu'une Collection désigne l'ensemble des objets appartenant à une classe (toutes les voitures en circulation).

Lorsque vous parlez d'acheter la 2CV rouge de vos rêves, vous évoquez une voiture concrète, bien spécifique. Vous créez une *instance* – on parle aussi d'une *occurrence* – de l'objet Voiture. Elle possède toutes les propriétés de la classe Voitures, mais ces propriétés sont attachées à des valeurs précises. La carrosserie est rouge, la vitesse maximale est de x km/h, etc. Vous pouvez maîtriser le comportement de votre voiture à l'aide des méthodes définies dans la classe Voitures (Accélérer, Freiner), mais l'effet précis de ces méthodes est étroitement lié aux propriétés de votre voiture. La puissance du moteur ne permet pas d'atteindre 200 km/h (mais vous pouvez décapoter !) ; les freins ne sont pas équipés du système ABS, il faut donc telle distance pour freiner, etc.

Un programme VBA peut ainsi créer une feuille de calcul Excel en appliquant la méthode Add (ajouter) à la collection WorkBooks et déterminer les propriétés de ce classeur (son nom, ses options de protection, le nombre de feuilles qui le composent, etc.)



Lorsque vous créez une instance d'objet, cet objet possède toutes les propriétés et méthodes définies dans la classe de l'objet. Ce principe essentiel de la programmation orientée objet est appelé instanciation.

Le grand intérêt de la programmation orientée objet est qu'il n'est pas utile de savoir comment fonctionne un objet pour l'utiliser. Lorsque vous achetez une voiture, vous n'avez pas besoin de savoir comment la carrosserie et le moteur ont été fabriqués, ni comment les différents composants sont assemblés, vous vous contentez de choisir un modèle, une couleur, etc. Il vous suffit de connaître les méthodes propres à la classe Voitures pour l'utiliser. Avec VBA, lorsque vous créez une instance d'un objet, vous en définissez les propriétés sans vous préoccuper de la façon dont celles-ci seront appliquées. Il en va de même pour les méthodes que vous utilisez pour maîtriser le comportement d'un objet. Lorsque vous tournez la clé de contact, le moteur de la voiture démarre, sans que vous ayez à vous soucier du détail des événements et des technologies mises en œuvre.

VBA permet, par exemple, de créer des interfaces graphiques pour vos programmes, en déposant simplement les objets dont vous avez besoin (cases à cocher, zones de texte, boutons de commandes, etc.), sur une feuille. Ces objets ont des comportements spécifiques que votre programme exploitera, sans que vous ayez besoin de vous soucier de leur mécanisme interne.

Application hôte et modèles d'objets

Lorsque vous développerez des programmes VBA, vous agirez sur des objets qui varieront en fonction des actions que vous souhaitez que votre programme exécute. Vous définirez et associerez ces objets de façon à créer une application complète. Là encore, l'analogie avec les objets de la vie courante est révélatrice. Les objets que nous utilisons sont généralement ordonnés selon leur fonction. Lorsque vous souhaitez vous laver, vous vous dirigez vers la salle de Bains ; il s'agit du lieu consacré à la toilette. Vous y trouvez un certain nombre d'objets tels que Savon, Gant de toilette, Dentifrice, Brosse à dents, etc. Vous utilisez le savon avec le gant de toilette, le dentifrice avec la brosse à dents, et vous pouvez faire une toilette complète.

Si vous souhaitez manger, c'est dans la cuisine que vous vous orienterez. Vous y trouverez quelques objets communs à ceux de la salle de bains (Lavabo, Robinet, Placard, etc.). Vous ne devriez cependant pas y trouver de brosse à dents, ni aucun des objets spécifiques à la toilette. Par contre, vous pourrez utiliser le four, ouvrir le frigo et utiliser tous les objets spécifiques de la cuisine.

Les applications du Pack Office sont comparables aux pièces de votre maison. Lorsque vous choisissez de développer un projet VBA, vous choisissez une *application hôte*. Il s'agit de l'application Office qui contient les objets sur lesquels vous souhaitez agir. C'est dans

cette application que vous développerez vos programmes, et c'est uniquement à partir de cette application qu'ils pourront être exécutés. Si vous souhaitez travailler sur des textes, vous choisirez d'entrer dans Word ; pour faire des calculs, vous savez que c'est dans Excel que vous trouverez les objets dont vous avez besoin ; Access sert au développement et au maniement des bases de données et PowerPoint, à la création de présentations.

Cependant, à l'image des pièces de votre maison, les applications Office ne sont pas hermétiques. Vous pouvez parfaitement vous préparer un plateau repas dans la cuisine et choisir de manger au lit. De façon semblable, des projets VBA évolués peuvent utiliser des objets de différentes applications Office. Un programme développé dans Excel peut utiliser des données stockées dans une base de données Access ou des objets Word pour imprimer un courrier qui accompagnera une facture, et envoyer un message Outlook de confirmation. Vous devez alors choisir une application hôte pour votre projet. Deux critères doivent la déterminer :

- Votre programme sera plus performant et plus simple à développer si l'application hôte est celle dans laquelle s'exécute l'essentiel des instructions du programme.
- La présence du programme dans l'application hôte doit être logique, et l'utilisateur final doit y avoir un accès facile puisque le programme ne pourra être exécuté qu'à partir de celle-ci.



Tous les projets développés dans cet ouvrage seront hébergés dans Excel. Pour accéder aux objets d'une application autre que l'application hôte, vous utiliserez la technologie Automation. L'accès aux objets d'une autre application est traité au Chapitre 6.

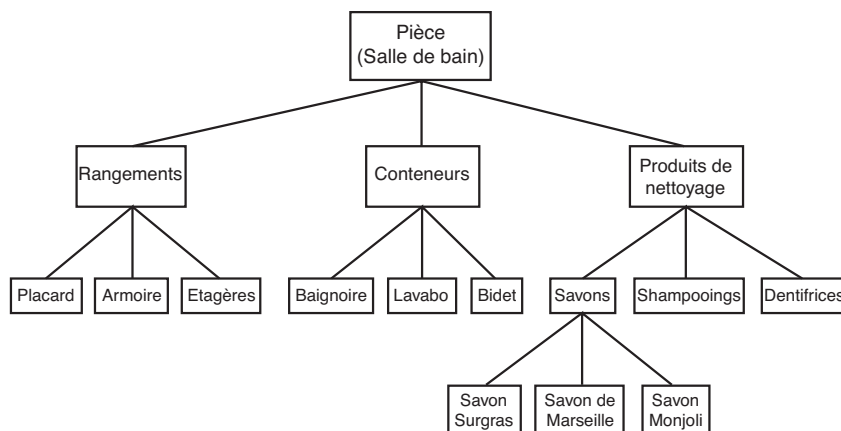
L'application est donc la pièce dans laquelle votre programme s'exécutera. Celle-ci est composée d'un certain nombre d'objets – constituant une *bibliothèque d'objets* – dont les rapports sont précisément définis. Les objets d'une application et les rapports qu'ils entretiennent sont représentés sous la forme d'un organigramme. Tout en haut de l'organigramme se trouve l'application (la pièce dans laquelle sont rangés tous les objets). Viennent ensuite les classes d'objets de premier niveau de l'application, auxquelles sont liés d'autres objets ou classes, et ainsi de suite. On appelle cette structure le *modèle d'objets* ou la *hiérarchie de classes* de l'application. La Figure 1.1 représente ce qui pourrait être un modèle d'objets sommaire de l'application Salle de bains.



La plupart des éléments d'Excel peuvent être manipulés dans Visual Basic pour Applications en tant qu'objets. Un classeur Excel, une feuille de ce classeur, une cellule ou une boîte de dialogue Rechercher sont des objets qui peuvent être manipulés dans un programme Visual Basic.

Figure 1.1

L'ensemble des objets d'une application est structuré selon un modèle d'objets qui en définit les rapports et la hiérarchie.



Au sommet du modèle se trouve la pièce – l'application. Tous les objets auxquels vous pouvez accéder y sont contenus. Si l'on établit un modèle d'objets pour l'ensemble des pièces de la maison, on retrouvera toujours l'objet **Pièce** au sommet du modèle. De la même façon, au sommet des modèles d'objets des applications Office, se trouve l'objet **Application**.

Viennent ensuite les classes situées immédiatement sous l'objet **Pièce**. Plus on progresse dans le modèle d'objets, plus les objets sont précis et donc spécifiques de la pièce ou de l'application. Par exemple, dans Excel, sous l'objet **Application** se trouve la collection (ou classe) **Workbooks** qui englobe tous les objets **Workbook**, c'est-à-dire tous les classeurs Excel ouverts. Sous l'objet **Workbook** se trouve la classe **Worksheets** qui englobe tous les objets **Worksheet** (toutes les feuilles de calcul) de l'objet **Workbook** désigné.

Astuce

*Pour accéder à l'aide en ligne des objets Excel, affichez l'Aide de Microsoft Visual Basic pour Applications à partir du menu "?", puis sélectionnez la commande **Référence du modèle d'objets Excel** (**Objets Microsoft Excel** si vous utilisez une version antérieure). Vous apprendrez à accéder à Visual Basic Editor au prochain chapitre.*

Notez que le fait que des objets appartiennent à des branches distinctes du modèle d'objets ne signifie pas qu'ils ne peuvent pas interagir. L'objet **Savon de Marseille** peut se trouver sur l'étagère, et vous pouvez utiliser la méthode **Déplacer** pour le mettre dans l'objet **Baignoire**, comme dans l'objet **Lavabo**.

Un objet peut englober d'autres objets. Un objet intégrant d'autres objets est qualifié de *conteneur*. C'est le cas de l'objet **Application**, mais c'est aussi vrai pour beaucoup d'autres objets du modèle d'objets d'Excel. Par exemple, un objet **Workbook** contient des

Pour accéder à un objet Excel, vous opérerez selon le même mode, c'est-à-dire en partant de l'objet situé le plus haut dans la hiérarchie d'objets et en progressant dans celle-ci jusqu'à atteindre l'objet voulu.

Le point est utilisé comme séparateur entre les différentes collections et objets que l'on rencontre avant d'atteindre l'objet voulu. La référence à un objet précis d'une collection se fait selon la syntaxe suivante :

```
Nom_Collection("Nom_Obj")
```

ou

Le code VBA permettant d'accéder à l'objet Dentifrice serait :

```
Piece.ProduitsNettoyants("Dentifrice").Prendre
```

La première partie du code permet d'accéder à l'objet Dentifrice ; l'expression identifiant un objet est appelée *référentiel d'objet*. La méthode Prendre est ensuite appliquée à cet objet afin de s'en saisir.

Le code Visual Basic permettant d'appeler la feuille de classeur Excel nommée "MaFeuille", et située dans le classeur "MonClasseur.xlsm" (à condition que celui-ci soit ouvert), serait :

```
Application.Workbooks("MonClasseur.xlsm").Sheets("MaFeuille").Activate
```

On accède à l'objet Workbook MonClasseur de la collection Workbooks (tous les classeurs ouverts), puis à la feuille nommée "MaFeuille" de la collection Sheets (toutes les feuilles de l'objet MonClasseur). Une fois le chemin d'accès à l'objet indiqué, on lui applique la méthode Activate pour l'activer.



Outre leur nom, les objets d'une collection sont identifiés au sein de la collection par une valeur d'indice représentant leur position dans la collection. Cette valeur peut être utilisée pour renvoyer un objet d'une collection, selon la syntaxe suivante :

```
Nom_Collection(IndexObjet)
```

où IndexObjet représente la position de l'objet dans la collection. L'instruction suivante :

```
Workbooks(2).Activate
```

active le classeur Excel apparaissant en deuxième position dans le menu Fenêtre.

Poursuivons l'analogie. Si vous vous trouvez déjà dans la salle de bains au moment où vous décidez de vous laver les dents, vous n'avez pas besoin d'y accéder. Si vous avez déjà le nez parmi les produits de toilette, il est inutile d'y faire référence.

De façon semblable, dans le code VBA, les objets de *niveau hiérarchique* supérieur à celui de l'objet que vous souhaitez atteindre peuvent parfois être ignorés. C'est toujours le cas pour l'objet Application. En effet, votre projet VBA étant stocké et donc exécuté à partir d'une application hôte, il est inutile de rappeler que vous êtes dans cette application.

L'expression :

```
Workbooks("MonClasseur.xlsm").Sheets("MaFeuille").Activate
```

suffit donc à activer la feuille intitulée "MaFeuille" du classeur nommé "MonClasseur.xlsm".

Selon le même principe, en cas d'absence de référentiel d'objets, la collection Sheets concerne le classeur actif. Si MonClasseur est le classeur actif, on peut donc se dispenser de toute référence à cet objet. On obtient alors l'instruction suivante :

```
Sheets("MaFeuille").Activate
```



Une petite finesse terminologique : les objets à proprement parler n'apparaissent jamais dans le code. Pour faire référence à un objet, on utilise une propriété qui appelle ou renvoie l'objet voulu. Dans les exemples précédents, Workbooks est une propriété de l'objet Application, qui renvoie tous les classeurs ouverts (la classe Workbooks). Sheets est une propriété de l'objet Workbook, qui renvoie toutes les feuilles de classeur (la classe Sheets) de cet objet.

Les propriétés

Revenons à l'analogie avec l'automobile et prenons la classe Voitures. Toutes les propriétés propres aux objets Voitures y sont définies. Les objets ou classes situés immédiatement sous Voitures dans le modèle d'objets appartiennent à la collection d'objets Voitures. En tant que tels, ils héritent de toutes les propriétés définies dans la classe Voitures.

Les propriétés peuvent être un attribut de l'objet ou un aspect de son comportement. Par exemple, les propriétés d'une voiture sont, notamment, sa marque, son modèle, l'état des pneus, l'activation ou non du moteur, etc. Les propriétés d'un document Word sont son modèle, son nom, sa taille, etc.

Les propriétés prennent des valeurs spécifiques qui distinguent les différents objets de la collection. La propriété Couleur d'un objet Voiture peut prendre la valeur Rouge, tandis que la même propriété d'un objet de la collection est attachée à la valeur Bleu.

Lorsque vous développerez des programmes VBA, vous exploiterez les propriétés d'un objet de deux façons :

- **En modifiant les valeurs attachées aux propriétés de l'objet.** Pour modifier un objet, il suffit de changer les valeurs de ses propriétés. Les propriétés dont les valeurs peuvent être modifiées sont dites en lecture-écriture.

Certaines propriétés ne peuvent être modifiées et sont dites en lecture seule. Vous pouvez, par exemple, modifier la propriété `Etat_du_moteur` (allumé ou éteint) d'un objet `Voiture`, mais non sa propriété `Marque`. Vous pouvez modifier le nombre de feuilles qui composent un classeur, mais non sa date de création.

- **En interrogeant les valeurs attachées aux propriétés d'un objet.** Les valeurs des propriétés peuvent être lues afin de connaître les spécificités de l'objet et d'orienter le comportement du programme. Par exemple, si la propriété `Marque` d'un objet `Voiture` est affectée à la valeur `BMW` et sa propriété `Contenu_Réservoir`, affectée à une valeur égale à 40 (litres), vous ferez un plein à 40 euros. Si les propriétés `Marque` et `Modèle` sont respectivement affectées aux valeurs `Citroën` et `2CV` et la propriété `Contenu_Réservoir`, à une valeur égale à 20, vous ne ferez qu'un plein à 20 euros.

De la même façon, un programme VBA peut exécuter des instructions tant que le nombre de classeurs ouverts est différent de zéro (`Workbooks.Count <> 0`).

Types de valeurs des propriétés

Les valeurs affectées aux propriétés d'un objet peuvent être de quatre types :

- une chaîne de caractères ;
- une valeur numérique ;
- une valeur booléenne ;
- une constante.

Chaînes de caractères

Une chaîne de caractères est une suite de caractères contigus – lettres, chiffres, espaces ou signes de ponctuation. Ces données sont aussi qualifiées de type *Chaîne* ou *String*. Une chaîne de caractères peut contenir jusqu'à environ deux milliards de caractères. En Visual Basic, les chaînes de caractères sont placées entre guillemets.

Les exemples suivants sont des chaînes de caractères :

- `"Paul"` ;
- `"1254"` ;
- `"Je suis une chaîne de caractères composée de 59 caractères"`.

Les chaînes de caractères sont interprétées en tant que caractères, et non en tant que valeur numérique. Autrement dit, la chaîne "1254" est interprétée comme la combinaison des caractères 1, 2, 5 et 4.

Par exemple, la propriété `Modèle` d'un objet `Voiture` est toujours une chaîne de caractères. Celle-ci ne peut être composée que de chiffres – par exemple "2000" –, sans que vous puissiez pour autant diviser cette valeur par un nombre quelconque.

Valeurs numériques

Une valeur numérique est une suite de chiffres. Elle peut être un nombre entier ou décimal, positif ou négatif.

Les exemples suivants sont des valeurs numériques :

- 0 ;
- 1 548 972 ;
- -1 245,4542 ;
- 100E4.



Le caractère E dans une variable numérique signifie "Exposant". Ainsi, la valeur numérique 100E4 est égale à 100×10^4 .

Les valeurs numériques sont interprétées comme des chiffres. Il peut s'agir de valeurs comme d'expressions conjuguant valeurs numériques et opérateurs arithmétiques (* / – +). Par exemple, les propriétés `Contenu_Réservoir` et `Consommation` d'un objet `Voiture` sont des valeurs numériques. La combinaison de ces propriétés permet de déterminer combien de kilomètres peuvent être parcourus avant la panne sèche, selon l'expression arithmétique :

`Kilomètres_Avant_Panne_Sèche = Contenu_Réservoir / Consommation`



Notez qu'une expression arithmétique peut être composée de nombres (100×25), de variables auxquelles sont affectées des valeurs numériques (`nombre1` $\times$ `nombre2`), ou d'une combinaison des deux (`nombre1` $\times$ 25). Les variables sont étudiées en détail au Chapitre 6.

Les valeurs numériques pouvant être affectées à une propriété varient avec les propriétés et les objets. Par exemple, dans Excel, la taille d'une police doit être comprise entre 1 et 409. Par conséquent, la valeur que peut prendre la propriété `Size` (taille) d'un objet `Font` (police)

d'Excel doit aussi être comprise entre ces deux valeurs. Dans le cas de l'objet *Voiture*, la propriété *Contenu_Réservoir* doit toujours être supérieure à 0, la valeur maximale dépendant d'autres spécificités de l'objet, telles que ses propriétés *Marque* et *Modèle*.

Valeurs booléennes

Certaines propriétés ne peuvent prendre que deux états : elles sont vérifiées ou elles ne le sont pas. Ces propriétés sont attachées à une valeur de type *Boolean*, ou valeur booléenne. Une valeur booléenne peut être *True* ou *False*.

La propriété *Moteur_Allumé* d'un objet *Voiture* est attachée à une valeur booléenne. La valeur *True* lui est affectée si le moteur de l'objet *Voiture* est allumé. Dans le cas contraire, la propriété *Moteur_Allumé* sera égale à *False*.

Comme vous le verrez au Chapitre 15, un classeur Excel gère une vingtaine de propriétés qui représentent ses options et son état à un moment donné. Nombre de ces propriétés acceptent une valeur de type *Boolean*. C'est par exemple le cas de la propriété *Saved* qui renvoie *True* si aucune modification n'a été apportée au document depuis son dernier enregistrement, et *False* dans le cas contraire.



*En Visual Basic, la valeur *True* peut être remplacée par – 1, et la valeur *False*, par 0. Cette pratique est cependant déconseillée, puisqu'elle rend la lecture du code moins aisée.*

Constantes

Les constantes sont des valeurs intégrées de VBA qui conservent toujours la même valeur. Lorsqu'une propriété accepte un nombre déterminé d'états, les valeurs représentant ces états sont souvent représentées sous forme de constantes. Celles-ci se présentent sous la forme d'une suite de lettres. Les constantes sont représentées sous forme de chaînes de caractères, mais correspondent en réalité à des valeurs numériques.

Les constantes intégrées désignent l'état de propriétés pour un objet spécifique. Chacune des applications Office possède ses propres constantes (puisque'elle possède ses propres objets). Cependant, certaines propriétés étant communes à toutes les applications de la suite Office, les constantes désignant leur état se retrouvent aussi dans toutes les applications Office. Les constantes intégrées de VBA commencent par deux lettres en minuscules indiquant l'application à laquelle appartient la constante. Le tableau ci-dessous reprend les préfixes des constantes VBA les plus courantes pour Microsoft Office :

vb	Visual Basic
wd	Word
xl	Excel
pp	PowerPoint
ac	Access
ol	Outlook
fp	FrontPage
bind	Classeur Office
fm	Feuilles Visual Basic

Lorsqu'une propriété accepte des constantes pour valeurs, le nombre de ces constantes est déterminé et correspond aux différents états que peut prendre la propriété. Par exemple, les clignotants d'une voiture peuvent accepter quatre états différents : désactivés, activés à droite, activés à gauche, position Warning (les clignotants droite et gauche activés). La propriété `Etat` d'un objet `Clignotant` pourrait donc accepter l'une des quatre constantes `Clignotant`, chacune correspondant à l'un de ces états possibles :

```
ClignotantAucun  
ClignotantDroite  
ClignotantGauche  
ClignotantWarning
```

Excel intègre un nombre important de constantes. Lorsqu'une commande Excel exige de l'utilisateur la sélection d'une option parmi plusieurs possibles, ces options sont généralement représentées sous forme de constantes en langage VBA. Par exemple, lorsque vous insérez une cellule dans une feuille de classeur (Insertion > Cellules), vous devez choisir entre les options Décaler les cellules vers la droite ou Décaler les cellules vers le bas. L'instruction VBA correspondante sera :

```
Selection.Insert(Shift)
```

où l'argument `Shift` est une des constantes `XlInsertShiftDirection` spécifiant à la méthode `Insert` la façon dont la cellule sera insérée. Il peut s'agir de la constante `xlShiftToRight` (les cellules seront décalées vers la droite) ou de la constante `xlShiftDown` (les cellules seront décalées vers le bas).



Les constantes sont la représentation textuelle de valeurs numériques. Chacune des constantes `Clignotant` correspond à une valeur numérique. La propriété `ClignotantWarning` pourrait, par exemple, correspondre à la valeur numérique 3. Vous pouvez indifféremment utiliser les constantes VBA ou les valeurs numériques auxquelles elles correspondent. Il est cependant conseillé d'utiliser les constantes, afin de faciliter la lecture du code. Si vous comprenez l'anglais, les constantes vous sembleront en effet plus parlantes que de simples chiffres.

Accéder aux propriétés

Pour modifier une propriété d'un objet, on utilise la syntaxe suivante :

Expression.Propriété = valeur

où *Expression* est une expression renvoyant un objet – un référentiel d'objet –, tel que cela a été décrit dans la section précédente. *Propriété* est le nom de la propriété que l'on souhaite modifier (toujours séparée de l'objet auquel elle se réfère par un point) et *valeur*, la valeur que vous souhaitez lui affecter.

Le type de la valeur (chaîne, valeur numérique, constante ou valeur booléenne) doit être adapté à la propriété. Si tel n'est pas le cas, le programme génère une erreur. Par exemple, la propriété `Contenu_Réservoir` d'un objet `Voiture` n'accepte qu'une valeur numérique ; vous ne pouvez pas lui affecter une chaîne de caractères.

Le Tableau 1.1 illustre différentes possibilités de modifier l'objet `Voiture` "MaVoiture" :

Tableau 1.1 : Pour modifier un objet, il suffit d'en changer les propriétés

<i>Syntaxe</i>	<i>Type de la valeur affectée</i>	<i>Conséquence pour l'objet Voiture</i>
<code>Voitures("MaVoiture").Immatriculation = "4444AB29"</code>	Chaîne de caractères	Une nouvelle immatriculation
<code>Voitures("MaVoiture").Moteur_Allume = True</code>	Valeur booléenne	Le moteur est allumé
<code>Voitures("MaVoiture").Contenu_Réservoir = 50</code>	Valeur numérique	Le réservoir contient 50 litres
<code>Voitures("MaVoiture").Clignotant.Etat = ClignotantWarning</code>	Constante	L'objet clignotant est en position Warning

Pour lire la valeur d'une propriété d'un objet, on stocke généralement cette valeur dans une variable, selon la syntaxe suivante :

variable = Expression.Propriété

L'instruction suivante passe la fenêtre active en mode d'affichage Aperçu des sauts de page, en définissant sa propriété `View` à `xlPageBreakPreview`.

`ActiveWindow.View = xlPageBreakPreview`

L'instruction suivante stocke dans la variable `TypeAffichage` la valeur représentant le type d'affichage en cours :

`TypeAffichage = ActiveWindow.View`

Les méthodes

Les méthodes représentent les actions qu'un objet peut exécuter. Tandis que les propriétés définissent un état, les méthodes déterminent un comportement. Les méthodes pouvant être appliquées à un objet dépendent étroitement de l'objet. Les objets de la classe *Voitures* supportent des méthodes telles que *Tourner*, *Freiner*, *Accélérer*, etc.

Cependant, certaines méthodes peuvent être communes à des objets différents, même si elles ont des conséquences différentes. Par exemple, la méthode *Ouvrir* peut s'appliquer aux objets *Porte*, *Coffre* ou *Cendrier* d'une voiture, comme à un objet *Porte* ou *Robinet* d'une maison. Certaines méthodes se retrouvent dans toutes les applications Office. C'est le cas pour toutes les méthodes correspondant à des commandes de menu communes aux applications. Par exemple, les méthodes *Open* (ouvrir) et *Close* (fermer) peuvent s'appliquer à un classeur Excel comme à un document Word, un formulaire Access ou encore une présentation PowerPoint.

Une méthode peut avoir des conséquences sur l'état de certaines propriétés de l'objet auquel elle s'applique, voire sur d'autres objets. Par exemple, si vous appliquez la méthode *Accélérer* à un objet *Voiture*, la valeur affectée à la propriété *Vitesse* de cet objet augmentera.

Si vous modifiez le contenu d'une cellule d'un classeur Excel, la taille de la cellule pourra être modifiée en conséquence. Si d'autres cellules sont liées par des formules à la cellule dont vous modifiez la valeur, leurs valeurs seront mises à jour en conséquence. Chaque fois que vous créez un nouveau classeur à l'aide de la méthode *Add*, la valeur de la propriété *Count* de la collection *Workbooks* (le nombre de classeurs ouverts) est incrémentée de 1. Chaque fois que vous fermez le classeur à l'aide de la méthode *Close*, la valeur de la propriété *Count* de la collection *Workbooks* est décrémentée de 1.

En outre, pour exécuter correctement une méthode, il est parfois nécessaire de modifier au préalable les propriétés de l'objet auquel elle s'applique. Par exemple, si vous souhaitez appliquer la méthode *Tourner* à un objet *Voiture*, vous devez auparavant modifier la propriété *Etat_Clignotant* de l'objet *Clignotant* de cette voiture.

La syntaxe permettant d'appliquer une méthode à un objet est :

`Expression.Méthode`

où *Expression* est une expression renvoyant un objet – un référentiel d'objet –, tel que cela a été décrit dans la section précédente. *Méthode* est le nom de la méthode que l'on souhaite exécuter (toujours séparée de l'objet auquel elle se réfère par un point).

Une méthode peut aussi s'appliquer à une collection d'objets. La syntaxe est alors :

Collection.Méthode

où *Collection* représente la collection d'objets sur laquelle on souhaite agir, et *Méthode*, une méthode commune aux objets de la collection.

Vous pouvez, par exemple, arrêter toutes les voitures de la collection Voitures, en leur appliquant la méthode Arrêter :

Voitures.Arrêter

Pour fermer tous les classeurs ouverts dans une session Excel, vous utiliserez l'instruction suivante :

Workbooks.Close

Cette syntaxe est aussi utilisée pour créer une occurrence d'un objet de la collection *Collection*. La méthode utilisée est alors généralement la méthode *Add* – l'équivalent Visual Basic de l'onglet Fichier. Par exemple, pour créer un nouveau classeur Excel, vous ferez appel à la collection *Workbooks* (*Workbooks.Add*).

Vous pouvez ensuite définir les propriétés de l'objet ainsi créé, comme nous l'avons vu dans la section "Les propriétés" de ce chapitre.

Les événements

Un événement est une action reconnue par un objet. La reconnaissance d'un événement par un objet permet de déclencher l'exécution d'un programme lorsque cet événement survient. On parle alors de *procédure événementielle*. Un clic de souris ou la frappe d'une touche sont des exemples d'événements pouvant être interprétés par un programme VBA.



Une procédure événementielle est une procédure attachée à un événement utilisateur tel qu'un clic de souris, la frappe d'une touche, l'activation d'une feuille de calcul, etc. La procédure s'exécute lorsque l'événement auquel elle est attachée est reconnu par l'application.

Les objets de la collection *Voitures* peuvent, par exemple, reconnaître l'événement *Choc*. Un objet *Voiture* peut être conçu pour que la détection de l'événement *Choc* entraîne l'ouverture de l'objet *Airbag*, autrement dit l'application de la méthode *Ouvrir* à cet objet.

Les événements s'utilisent essentiellement avec les contrôles de formulaires que vous développerez et avec les objets. Vous apprendrez à exploiter les événements utilisateur affectant

un formulaire aux Chapitres 13 et 14. Les feuilles de calcul, les graphiques, les classeurs et l'application Excel gèrent aussi des événements. Vous apprendrez à créer des procédures événementielles pour ces objets au Chapitre 15.

Les fonctions

Les fonctions servent à renvoyer une information, selon les éléments qui leur sont fournis. Le type de l'information renvoyée varie d'une fonction à l'autre. Il peut s'agir d'une chaîne de caractères, d'une valeur numérique, booléenne, de type Date, etc. Visual Basic intègre un certain nombre de fonctions que vous pouvez exploiter directement. Par exemple, la fonction Asc renvoie le code ASCII du caractère sélectionné, tandis que la fonction Int renvoie la partie entière d'un nombre. Certaines fonctions sont particulièrement utiles. C'est le cas de la fonction MsgBox qui permet d'afficher une boîte de dialogue contenant des boutons tels que Oui, Non, Annuler, etc., et qui renvoie une valeur reflétant le choix de l'utilisateur.

Vous pouvez aussi créer vos propres fonctions qui traiteront les valeurs qui leur seront passées pour renvoyer une valeur ensuite utilisée par le programme. Dans le cas d'un objet Voiture, vous pouvez créer une fonction Coût_Plein qui exploitera les propriétés Contenu_Réservoir et Contenance_Réservoir de l'objet, ainsi qu'une variable représentant le prix de l'essence, pour renvoyer une valeur correspondant au coût d'un plein. Lorsque vous créez des fonctions VBA pour Excel, celles-ci sont accessibles pour l'utilisateur final comme n'importe quelle fonction Excel intégrée.

Les fonctions ont généralement besoin de *paramètres* ou *arguments*. Si les *arguments* obligatoires d'une fonction ne lui sont pas passés au moment de l'appel, une erreur est générée. Dans le cas précédent, trois paramètres de type numérique doivent être passés à la fonction Coût_Plein pour qu'elle s'exécute correctement : le contenu du réservoir, sa contenance et le prix de l'essence.

Le modèle d'objets d'Excel

Excel est l'application Office qui supporte VBA depuis le plus longtemps et son modèle d'objets est le plus mûr. Excel offre de multiples possibilités de personnalisation au programmeur.

Les objets les plus importants sont présentés dans le Tableau 1.2. Le Listing 1.1 présente des exemples d'instruction VBA utilisant ces objets. L'essentiel de ces exemples a été généré à l'aide de l'Enregistreur de macro, sans qu'il soit nécessaire d'écrire du code.

Tableau 1.2 : Les objets clés du modèle d'objets d'Excel

<i>Collection (objet)</i>	<i>Description</i>
Objets de niveau Application	
Add-ins (Add-in)	L'ensemble des macros complémentaires, chargées ou non. Accessibles dans la boîte de dialogue Macros complémentaires (Outils > Macros complémentaires).
Dialogs (Dialog)	Les boîtes de dialogue prédéfinies d'Excel.
LanguageSettings	Renvoie des informations sur les paramètres de langue utilisés dans l'application.
Names (Name)	L'ensemble des objets Name de niveau Application. Un objet Name représente un nom défini pour une plage de cellules nommée.
Windows (Window)	L'ensemble des fenêtres disponibles (accessibles <i>via</i> le menu Fenêtre).
Workbooks (Workbook)	L'ensemble des classeurs ouverts.
Worksheetfunction	On utilise l'objet Worksheetfunction pour accéder aux fonctions de feuilles de calcul à partir de VBA. Faites suivre la propriété Worksheetfunction d'un point, puis du nom de la fonction et de ses arguments entre parenthèses.
Objets de l'objet Workbook	
Charts (Chart)	L'ensemble des feuilles graphiques de l'objet Workbook.
Names (Names)	L'ensemble des objets Name pour le classeur spécifié.
Styles (Style)	L'ensemble des styles disponibles dans un classeur. Il peut s'agir d'un style défini par l'utilisateur ou d'un style prédéfini, tel que les styles Millier, Monétaire ou Pourcentage (Format > Styles).
Windows (Window)	L'ensemble des fenêtres pour le classeur spécifié.
Worksheets (Worksheet)	L'ensemble des feuilles de calcul de l'objet Workbook désigné.
Objets de l'objet Worksheet	
Names (Name)	L'ensemble des objets Name pour la feuille de calcul spécifiée.
Range	Une cellule, une ligne, une colonne ou une plage de cellules, contiguës ou non, une plage de cellules 3D.
Comments (Comment)	L'ensemble des commentaires pour l'objet Worksheet désigné.
HPageBreaks (HPageBreak)	Les sauts de page horizontaux de la feuille de calcul.
VPageBreaks (VPageBreaks)	Les sauts de page verticaux de la feuille de calcul.
Hyperlinks (Hyperlink)	L'ensemble des liens hypertexte de la feuille de calcul.
Scenarios (Scenario)	Les scénarios de la feuille de calcul.

<i>Collection (objet)</i>	<i>Description</i>
OLEObjects (OLEObject)	Les objets incorporés ou liés et les contrôles ActiveX de la feuille.
Outline	Le plan de la feuille de calcul.
PageSetup	Les options de mise en page de la feuille.
QueryTables (QueryTable)	Les tables de requête de la feuille.
PivotTables (PivotTable)	Les tableaux et les graphiques croisés dynamiques.
ChartObjects (ChartObject)	Les graphiques incorporés de la feuille de calcul spécifiée.
Objets de l'objet Range	
Areas	Les plages de cellules contiguës à l'intérieur d'une sélection.
Borders (Border)	Les bordures d'un objet Range. La collection Borders regroupe toujours quatre objets Border, représentant les quatre bordures de l'objet Range désigné.
Font	Les attributs de police de caractères de l'objet Range spécifié.
Interior	L'intérieur de l'objet Range.
Characters	L'ensemble des caractères contenus par l'objet Range.
Name	Le premier nom dans la liste des noms de la plage de cellules précisée.
Style	Le style de l'objet Range désigné.
FormatConditions (FormatCondition)	L'ensemble des mises en forme conditionnelles de l'objet Range.
Hyperlinks (Hyperlink)	L'ensemble des liens hypertexte de l'objet Range.
Validation	La validation des données pour la plage de cellules précisée.
Comment	Le commentaire de cellule pour l'objet Range désigné.

Listing 1.1 : Exemples d'utilisation des objets Excel

```
'activation du classeur Classeur1
Windows("Classeur1").Activate
'-----
'sauvegarde du classeur actif
ActiveWorkbook.Save
'nouveau classeur
Workbooks.Add
'nouveau classeur fondé sur le modèle MonModele.xlt
'-----
```

```
'affectation du nom MaPlage à la palge de cellule A1:C20 de la feuille MaFeuille
ActiveWorkbook.Names.Add Name:="hop", RefersToR1C1:="=Feuil1!R5C2:R12C3"
Names.Add Name:="MaPlage", RefersTo:="=MaFeuille!$a$1:$c$20"
'ajout d'un graphique
Charts.Add
'affectation du type Histogramme empilé au graphique actif
ActiveChart.ChartType = xlColumnStacked
'définition de la source de données du graphique actif
ActiveChart.SetSourceData Source:=Sheets("Feuil1").Range("C6:E10"),
➡PlotBy :=xlColumns
'définition de l'emplacement du graphique actif
ActiveChart.Location Where:=xlLocationAsObject, Name:="Feuil1"
'définition des titres du graphique actif
With ActiveChart
    .HasTitle = False
    .Axes(xlCategory, xlPrimary).HasTitle = False
    .Axes(xlValue, xlPrimary).HasTitle = False
End With
'-----
'sélection de la feuille Feuil1 du classeur actif
Sheets("Feuil1").Select
'affectation du nom Graphique à la feuille Feuil1
Sheets("Feuil1").Name = "Graphique"
'suppression des feuilles sélectionnées
ActiveWindow.SelectedSheets.Delete
'-----
'ajout d'un commentaire à la cellule D2 de la feuille active
Range("D2").AddComment
'le commentaire n'est pas rendu visible
Range("D2").Comment.Visible = False
'définition du texte du commentaire de la cellule D2
Range("D2").Comment.Text Text:="Excellent !"
'-----
'ajout du contrôle ActiveX Calendar à la feuille active
ActiveSheet.OLEObjects.Add(ClassType:="MSCAL.Calendar", Link:=False, _
    DisplayAsIcon:=False).Select
'-----
'affectation du format monétaire US à la plage sélectionnée
Selection.NumberFormat = "#,##0.00 $"
'définition des attributs de police de la plage sélectionnée
With Selection.Font
    .Name = "Arial"
```

```
.FontStyle = "Gras"  
.Size = 8  
.ColorIndex = 46  
End With  
'coloriage de l'intérieur de la plage sélectionnée  
With Selection.Interior  
.ColorIndex = 6  
.Pattern = xlSolid  
End With
```




2

Premières macros

Au sommaire de ce chapitre

- Créer une macro GrasItalique
- Différentes méthodes d'enregistrement
- Écrire la macro
- Créer une macro Titre_WordArt
- Stocker des macros

L'enregistrement de macros constitue certainement le meilleur apprentissage de Visual Basic pour Applications. Les commandes de l'application hôte accessibles par les menus, les barres d'outils ou les raccourcis clavier, le déplacement (à l'aide du clavier ou de la souris) dans un classeur et la modification de ce dernier peuvent être enregistrés dans une macro. Il suffit simplement de déclencher l'Enregistreur de macro et d'exécuter ces commandes, sans qu'il soit nécessaire d'écrire la moindre ligne de code. Cette méthode permet ensuite de répéter autant de fois que vous le souhaitez la série d'instructions ainsi mémorisées, en exécutant simplement la macro. Lorsque la série de commandes est enregistrée dans une macro, vous pouvez en visualiser le *codage* dans la fenêtre Code de Visual Basic Editor. Vous découvrirez ainsi la structure et la syntaxe des programmes VBA par la pratique.



Le code est le texte, écrit dans le langage de programmation, constituant le programme. Le codage désigne le fait de générer du code, soit en utilisant l'Enregistreur de macro, soit en l'écrivant directement dans la fenêtre de code de Visual Basic Editor.

À travers des exemples simples, ce chapitre vous initiera à l'enregistrement et à la création de macros. Vous créerez une première macro, puis en améliorerez très simplement la fonctionnalité. Vous verrez que l'enregistrement de macros est relativement souple, et que la création d'une macro offre plusieurs possibilités, plus ou moins efficaces et plus ou moins rapides. Vous apprendrez rapidement à utiliser l'une ou l'autre des méthodes disponibles (voire à les combiner), en fonction de l'objet de votre macro.

Créer une macro GrasItalique

Lorsque vous souhaitez enrichir le contenu d'une cellule d'attributs de caractères, une solution consiste à choisir le Format de cellule du bouton Format (onglet Accueil), et à sélectionner l'onglet Police. On définit ensuite les attributs voulus et l'on valide en cliquant sur OK. Nous utiliserons ici cette méthode pour créer une macro enrichissant la cellule ou la plage de cellules active des attributs gras et italique.

Cette macro est fort simple, puisque simplement composée de deux commandes, mais permettra de découvrir la façon dont les programmes VBA sont structurés. Le but de ce chapitre est de vous initier aux différentes méthodes de création et d'optimisation de macros. Prenez donc le temps de le lire dans sa totalité ; les principes acquis seront valables pour l'ensemble des macros que vous créerez par la suite, et ce quel que soit leur niveau de complexité.

Afficher l'onglet Développeur

Avant toute chose, vous devez afficher l'onglet Développeur dans le ruban pour accéder aux fonctions de programmation VBA. Cliquez sur l'onglet Fichier du ruban, puis sur la commande Options. Dans la fenêtre Options Excel, sélectionnez Personnaliser le ruban. Cochez ensuite la case Développeur de la liste Onglets principaux (voir Figure 2.1), puis validez. L'onglet Développeur apparaît sur le ruban.



Avec Office 2007, cliquez sur le bouton Office, situé dans l'angle supérieur gauche de la fenêtre Excel ; dans la fenêtre qui s'affiche, cliquez sur le bouton Options Excel. Cochez ensuite la case Afficher l'onglet Développeur dans le ruban, puis validez. Dans les versions 2007, l'accès aux macros se fait via la commande Macros du menu Outils.

Figure 2.1

Activez l'onglet Développeur pour accéder aux fonctions de programmation du logiciel.

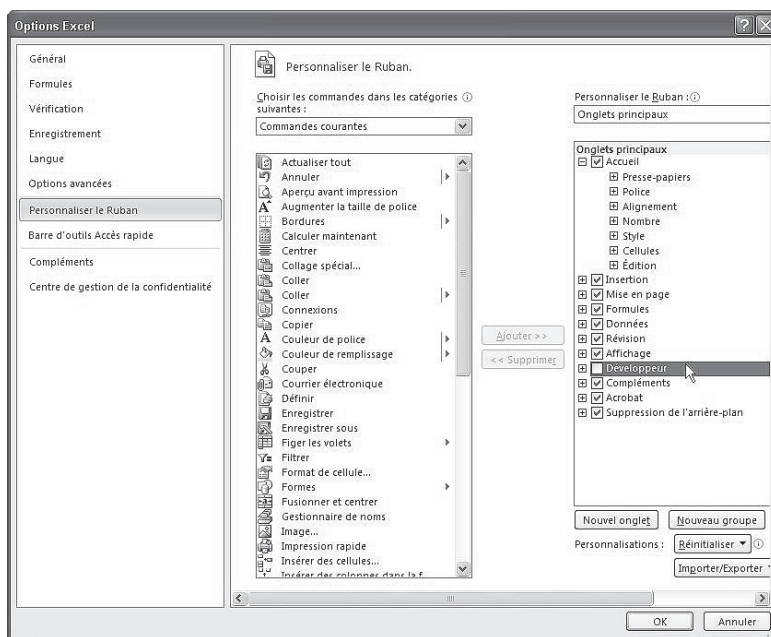
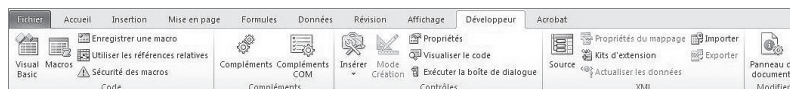


Figure 2.2

L'onglet Développeur est maintenant accessible sur le ruban.



Démarrer l'enregistrement

Avant de commencer l'enregistrement de la macro GrasItalique, sélectionnez une cellule à laquelle vous attribuerez les formats de caractères voulus.

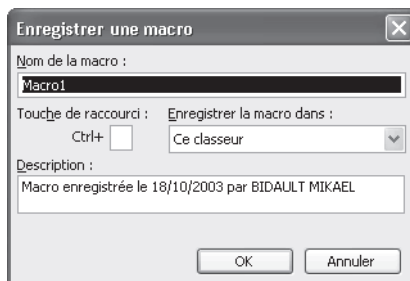
1. Cliquez sur le bouton Enregistrer une macro du groupe Code de l'onglet Développeur afin de lancer l'enregistrement de la macro.



Dans les versions antérieures à 2007 d'Office, les fonctions VBA sont accessibles via la commande Macro du menu Outils (ici Outils > Macro > Nouvelle macro). Ainsi, lorsque nous ferons référence à l'onglet Développeur du menu ruban, orientez-vous vers la commande Macros du menu Outils si vous utilisez une version d'Excel utilisant les menus classiques.

Figure 2.3

La boîte de dialogue
Enregistrer une macro.



2. Par défaut, la zone Nom de la macro indique Macro1. Remplacez ce nom par GrasItalique.



Il est plus rapide d'enregistrer une macro sous le nom que lui attribue Excel par défaut. Cependant, si vous enregistrez plusieurs macros, celles-ci deviendront rapidement indiscernables. Attribuez un nom représentatif à vos macros et entrez une brève description de leur fonction dans la zone Description, vous n'aurez ainsi aucun problème pour les distinguer.

3. Dans la zone Enregistrer la macro dans, choisissez Classeur de macros personnelles.
4. Dans la zone Description, tapez une brève description de la macro, en conservant la date de création et, éventuellement, le nom du créateur.



Lorsque vous enregistrez une macro, la date de son enregistrement et le nom d'utilisateur déclaré pour l'application apparaissent dans la zone Description sous la forme "Macro enregistrée le Date par Utilisateur". Vous retrouverez les

mêmes indications dans la fenêtre de Code de la macro. Si vous partagez vos macros avec d'autres utilisateurs, il peut être utile de conserver le nom du créateur afin de les identifier rapidement. De même, conserver la date de création d'une macro permettra de la situer plus facilement. Cette indication se révélera très utile si vous devez mettre à jour des macros.

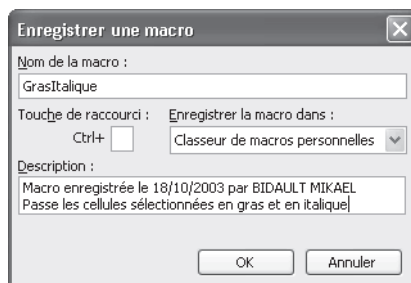
5. L'intérêt de la macro GrasItalique réside dans le gain de temps qu'elle apporte à l'utilisateur. L'attribution d'un raccourci clavier lui donnera donc toute son efficacité.

Placez le curseur dans la zone de texte Touche de raccourci et saisissez une lettre qui, combinée à la touche Ctrl, sera affectée à l'exécution de la macro GrasItalique (dans notre exemple, la combinaison Ctrl+B). Vous pouvez aussi maintenir la touche Maj enfoncée de façon à affecter à votre macro une combinaison Ctrl+Maj+Lettre.

La boîte de dialogue Enregistrer une macro doit maintenant se présenter comme à la Figure 2.4.

Figure 2.4

*La boîte de dialogue
Enregistrer une macro
complétée.*



Lorsque vous attribuez un raccourci clavier à une macro, aucune indication ne vous est fournie quant à l'affectation ou non de ce raccourci à une commande. Si le raccourci choisi était déjà affecté à une commande Excel, il sera réattribué à la macro sans que vous en soyez notifié. Veillez donc à ne pas attribuer à votre macro un raccourci clavier déjà utilisé par Excel. Et ce particulièrement si d'autres utilisateurs sont amenés à utiliser vos macros. Ils risqueraient en effet d'exécuter involontairement la macro en pensant utiliser le raccourci clavier d'une commande Excel.

6. Enfin, cliquez sur OK.

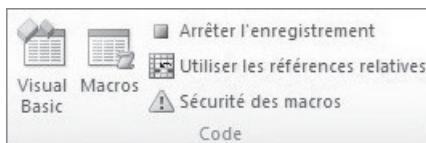
Le libellé du bouton Enregistrer une macro devient Arrêter l'enregistrement, indiquant que la macro est en cours d'enregistrement.



Dans les versions d'Excel antérieures à Office 2007, la barre d'outils Arrêt de l'enregistrement, simplement composée de deux boutons, s'affiche à l'écran. Le bouton Arrêter l'enregistrement permet d'interrompre l'enregistrement de la macro, tandis que le bouton Référence relative détermine l'enregistrement de vos déplacements dans la feuille Excel – ce sujet est abordé plus loin dans ce chapitre.

Figure 2.5

La commande Enregistrer une macro devient Arrêt de l'enregistrement, précisant que l'enregistrement a commencé.



Enregistrer les commandes de la macro

Comme nous l'avons dit au chapitre précédent, la création d'une macro simple ne nécessite pas la moindre ligne d'écriture. Il suffit d'exécuter les commandes qui la composent après avoir activé l'Enregistreur de macro : l'application hôte se charge de convertir les commandes exécutées en langage Visual Basic.

Pour enregistrer la macro *GrasItalique* :

1. Activez l'onglet Accueil du ruban puis cliquez sur le bouton Format, situé dans l'angle inférieur droit du groupe Cellules, et sélectionnez la commande Format de cellule. Vous pouvez également utiliser le raccourci clavier Ctrl+Maj+F.
2. Dans la zone Style, sélectionnez Gras italique, puis cliquez sur OK.
3. Les commandes de la macro *GrasItalique* sont enregistrées. Cliquez sur le bouton Arrêter l'enregistrement de l'onglet Développeur.



Si vous sélectionnez une cellule après avoir déclenché l'Enregistreur de macro, cette manipulation sera enregistrée. Par conséquent, la macro appliquera la mise en forme Gras Italique à cette cellule, et non aux cellules actives au moment de son exécution.

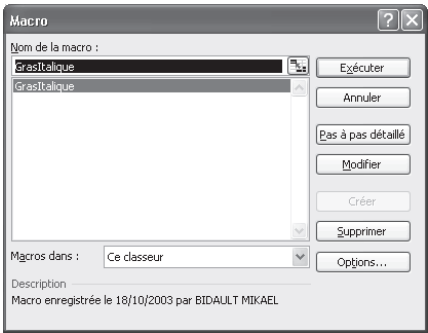
Exécuter la macro

L'exécution d'une macro est fort simple. Pour exécuter la macro *GrasItalique*, vous pouvez procéder de manière classique, ou – et c'est là que réside son intérêt – utiliser le raccourci clavier que nous lui avons attribué.

La boîte de dialogue Macro

- 1. Activez l’onglet Développeur du ruban, puis cliquez sur le bouton Macros du groupe Code. La boîte de dialogue Macro s’affiche.

Figure 2.6
La boîte de dialogue Macro.



- 2. Dans la liste des macros disponibles, sélectionnez GrasItalique. Le nom de la macro s’affiche dans la zone Nom de la macro.
- 3. Cliquez sur le bouton Exécuter. La boîte de dialogue Macro disparaît automatiquement et les cellules sélectionnées s’enrichissent des attributs Gras et Italique (voir Figure 2.7).

Si la procédure d’exécution que vous venez de mettre en œuvre convient à certaines macros plus complexes et d’un usage moins fréquent, elle ne présente pas d’intérêt pour la macro GrasItalique puisqu’elle nécessite plus d’opérations pour l’utilisateur qu’elle n’en exécute.

Figure 2.7
Les cellules sélectionnées
après exécution de la macro.

	A	B	C	D	E	F	G
1	<u>Répartition des représentants par départements</u>						
2							
3	Valérie Marie	Hervé Dubeuf	Hélène Legrand	Ludovic Bidault	Noel CAPLIER	Mathias Fried	Ma Duch
4	75	10	09	18	01	04	03
5	92	21	12	22	03	05	08
6	95	25	16	28	07	06	14
7		39	17	29	15	11	21
8		51	19	35	26	13	56
9		52	23	36	38	30	61
10		54	24	37	42	34	63
11		55	31	41	43	48	76
12		57	32	44	63	66	81
13		58	33	49	69	83	91

Le raccourci clavier

Sélectionnez le texte voulu, puis tapez le raccourci clavier attaché à la macro (Ctrl+B). En un clin d’œil les cellules sélectionnées se sont enrichies des attributs de caractères voulus.

Structure de la macro

Lors de l'enregistrement de la macro, les actions que vous avez effectuées ont été converties en langage Visual Basic. Pour en visualiser la syntaxe :

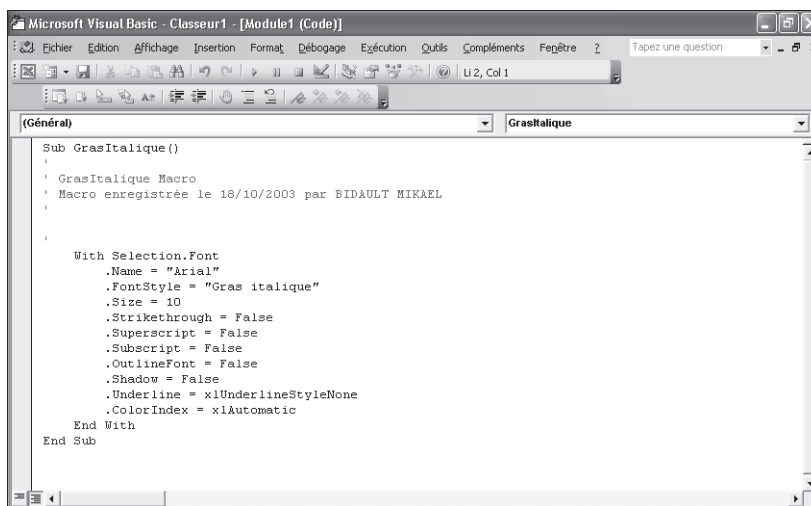
1. Activez l'onglet Développeur, puis cliquez sur le bouton Macros. Dans la boîte de dialogue Macro, sélectionnez la macro GrasItalique. Le nom sélectionné s'affiche dans la zone Nom de la macro.
2. Cliquez sur le bouton Modifier. Visual Basic Editor, l'environnement de développement intégré d'Office, s'ouvre sur la fenêtre Code de votre macro (voir Figure 2.8).

Info

Lorsque vous tentez de modifier la macro, si Excel affiche le message "Impossible de modifier une macro dans un classeur masqué...", vous devez afficher le fichier PERSONAL.XLSB. Sélectionnez l'onglet Affichage, puis cliquez sur le bouton Afficher du groupe Fenêtre. Si vous utilisez une version antérieure à Excel 2007, sélectionnez la commande Afficher du menu Fenêtre. Sélectionnez ensuite le fichier PERSONAL.XLSB (Perso.xls dans les anciennes versions d'Excel) dans la fenêtre qui s'affiche et validez.

Figure 2.8

La fenêtre Code de Visual Basic Editor permet de visualiser et de modifier le code d'une macro.



Examinons de plus près le texte de la macro GrasItalique. Il commence par l'instruction :

```
Sub GrasItalique()
```

et se termine par l'instruction :

```
End Sub
```

Ces deux instructions encadrent systématiquement toute macro enregistrée. Sub est l'abréviation de *subroutine* qui signifie *sous-routine* ou *sous-programme*. Les macros sont en effet des sous-programmes de l'application hôte. GrasItalique() est le nom de la sous-routine – c'est-à-dire le nom de la macro. End Sub indique la fin de la macro. Ces instructions sont indispensables au fonctionnement de toute macro ; cependant, vous n'aurez pas à vous en soucier : lorsque vous enregistrez une macro, les instructions qui la composent sont systématiquement encadrées par celles-ci.

Directement placées derrière l'instruction Sub *NomMacro*, des lignes de commentaires reprennent les informations que contenait la zone Description lors de l'enregistrement de la macro (voir Figure 2.3). En l'occurrence le nom du créateur et la date d'enregistrement de la macro, ainsi que le texte descriptif de la macro.



Les commentaires sont des indications ajoutées dans le code d'un programme et destinées à en faciliter la reconnaissance et/ou la compréhension.

Certains éléments du code apparaissent en couleur. Cette mise en valeur permet de distinguer aisément les éléments constitutifs du code. Par défaut, Visual Basic Editor applique la couleur verte aux commentaires et la couleur bleue aux mots clés du langage.



Un mot clé est un mot ou un symbole reconnu comme élément du langage de programmation Visual Basic. Il peut s'agir d'une structure de contrôle, d'une fonction ou de tout autre élément du langage indépendant du modèle d'objets de l'application hôte. Les structures de contrôle sont des instructions qui permettent de diriger le comportement d'une macro (par exemple, répéter une opération en boucle, n'effectuer une instruction que dans un contexte spécifique). Vous apprendrez à utiliser les structures de contrôle de Visual Basic au Chapitre 7.

Entre les instructions Sub GrasItalique() et End Sub se trouvent les instructions qu'exécutera la macro :

```
With Selection.Font
    .Name = "Arial"
    .FontStyle = "Gras italique"
    .Size = 10
    .Strikethrough = False
    .Superscript = False
    .Subscript = False
    .OutlineFont = False
    .Shadow = False
```

```

.Underline = xlUnderlineStyleNone
.ThemeColor = xlThemeColorLight1
.TintAndShade = 0
.ThemeFont = xlThemeFontMinor

End With

```



Si vous utilisez une version antérieure à Excel 2007, les trois lignes ThemeColor, TintAndShade et ThemeFont de code la ligne .ColorIndex = xlAutomatic sont remplacées par une seule ligne de code :

```
.ColorIndex = xlAutomatic
```

Cette différence vient de nouvelles fonctionnalités apparues avec Excel 2007.

Il s'agit des commandes effectuées lors de l'enregistrement : ces lignes indiquent à la macro les actions à accomplir. Leur structure peut vous dérouter, mais vous vous y habituerez rapidement :

- L'expression Selection.Font indique à la macro qu'il s'agit d'appliquer un format de police aux cellules sélectionnées :
 - Selection est une propriété qui renvoie un objet Selection représentant la sélection en cours dans le document actif. Lorsque vous enregistrerez des macros, vous verrez que certains objets, et les propriétés qui leur sont associées, sont très usités. C'est le cas de Selection, qui apparaît dans le code d'une macro Excel chaque fois qu'une opération (format de police, dimensions, définition d'une catégorie de données, etc.) est effectuée sur une plage de cellules sans que celle-ci soit définie auparavant.



Le code d'un projet VBA reflète la hiérarchie d'objets (ou modèle d'objets) de l'application hôte. Des propriétés sont utilisées pour appeler des objets spécifiques. Pour un rappel de ces concepts, reportez-vous au Chapitre 1.

- Font indique à la macro qu'il s'agit d'appliquer un format de police à l'objet Selection (les cellules sélectionnées).
- Les instructions With et End With encadrent l'ensemble des propriétés de l'objet Font. Comme tout mot clé, elles apparaissent en bleu dans la fenêtre de code. Lorsque, durant l'enregistrement d'une macro, vous faites appel à une boîte de dialogue dans laquelle plusieurs options sont définies, cette structure est utilisée pour coder l'ensemble des options de la boîte de dialogue, selon la syntaxe suivante :

```

With Objet
    Propriétés de l'objet
End With

```



Le verbe Coder désigne la transcription d'actions propres à l'application dans un langage de programmation déterminé.

- Chaque ligne située entre les instructions With Selection.Font et End With correspond à une option de l'onglet Police lors de l'enregistrement de la macro. Il s'agit des propriétés de l'objet Font. Remarquez que les propriétés sont toujours précédées d'un point.

À chaque propriété est affectée une *valeur*. Elle indique l'état de cette option lors de l'enregistrement de la commande. Cette valeur peut être :

- **False** ou **True** (valeur booléenne). Indiquent respectivement que l'option n'était pas cochée (faux) ou qu'elle l'était (vrai). `Superscript = False` indique ici que la case à cocher Exposant n'était pas validée lors de l'enregistrement de la macro.



Vous pouvez aussi utiliser les valeurs -1 et 0 à la place de True et False. Par exemple, l'expression `.Superscript = False` pourra être remplacée par `.Superscript = -1`.

- **Une chaîne de caractères.** Lorsqu'une propriété est attachée à une chaîne de caractères, cette valeur est placée entre guillemets. `Name = "Arial"` indique le nom de la police en cours dans la boîte de dialogue Police lors de l'enregistrement de la macro.
- **Une valeur numérique.** Les valeurs possibles varient d'une propriété à l'autre. `Size = 10` indique ici que le corps de la police est de 10 points. Dans Excel, cette valeur doit être définie entre 1 et 409.
- **Une constante.** Il s'agit d'une valeur prédéfinie qui permet de paramétrer une propriété. Par exemple, la propriété `Underline` définit le type de soulignement appliqué à la police ou à la plage. Sa valeur correspond à l'état de l'option Soulignement lors de l'enregistrement de la macro. Elle est ici attachée à la constante `xlUnderlineStyleNone` qui correspond à l'option Aucun de la zone de liste déroulante Soulignement. Il existe une constante `xlUnderlineStyle` spécifique pour chaque option de la zone de liste déroulante Soulignement (`xlUnderlineStyleDouble` pour Soulignement double, `xlUnderlineStyleSingle` pour Soulignement simple, etc.).

Le Tableau 2.1, en présentant à quelles options de la boîte de dialogue Format de cellule (onglet Police) les propriétés de l'objet Font sont associées, vous aidera à comprendre comment les actions que vous effectuez après avoir activé l'Enregistreur de macro sont codées par Visual Basic pour Applications.

Tableau 2.1 : Les propriétés de l'objet *Font* d'Excel

<i>Propriété</i>	<i>Format de cellule (onglet Police)</i>	<i>Valeurs autorisées</i>
Name	Zone de texte Police	Chaîne de caractères correspondant au nom d'une police disponible dans la zone de liste modifiable ¹ .
FontStyle	Zone de texte Style	Chaîne de caractères correspondant à l'option sélectionnée dans la zone de liste ¹ .
Size	Zone de texte Taille	Valeur numérique représentant le corps de la police. Cette valeur peut être comprise entre 1 et 409 ¹ .
Strikethrough	Case à cocher Barré	True (barré) ou False (non barré) ¹ .
SuperScript	Case à cocher Exposant	True (mise en forme exposant) ou False (pas de mise en forme exposant) ¹ . Les attributs Exposant et Indice ne pouvant être appliqués à une même sélection, lorsque vous affectez la valeur True à la propriété SuperScript, la propriété SubScript prend la valeur False ² .
SubScript	Case à cocher Indice	True (mise en forme indice) ou False (pas de mise en forme indice) ² . Les attributs Indice et Exposant ne pouvant être appliqués à une même sélection, lorsque vous affectez la valeur True à la propriété SubScript, la propriété SuperScript prend la valeur False ² .
OutlineFont et Shadow	[Aucune correspondance]	True ou False (sans effet). Ces propriétés indiquent respectivement si la police possède une mise en forme Relief et Ombré. Elles ne correspondent à aucune option de la boîte de dialogue Format de cellule, mais ont été conservées comme propriétés de l'objet Font d'Excel. Elles sont sans effet sur la police.
Underline	Zone de liste déroulante Soulignement	Une des cinq constantes xlUnderlineStyleNone représentant les cinq types de soulignement disponibles dans Excel ¹ .
ThemeColor, TintAndShade et ThemeFont	Onglet Remplissage	Respectivement une des constantes xlThemeColor (la couleur de motif), une valeur numérique comprise entre -1 et 1 représentant la teinte appliquée à cette couleur (de sombre à lumineux) et une des constantes xlThemeFont qui correspond à la police du thème.

1. Si vous interrogez la valeur d'une propriété pour une plage contenant des cellules dont les attributs correspondants sont différents, la valeur Null sera renvoyée. Par exemple, si vous interrogez la valeur de la propriété Name de l'objet Font d'une plage de cellules contenant à la fois des cellules en police Arial et d'autres en police Times, la valeur Null sera renvoyée.
2. Notez que cet état est le reflet de ce qui se passe dans la boîte de dialogue Police. Vous ne pouvez pas en effet cocher à la fois l'option Indice et l'option Exposant.

Comme le montre le tableau précédent, les actions exécutées sont codées selon des principes récurrents auxquels l'enregistrement de macros vous familiarisera.

La macro `GrasItalique` ouvre donc (virtuellement) la boîte de dialogue `Format de cellule` sur l'onglet `Police` et y définit les options telles qu'elles l'ont été lors de l'enregistrement. Elle applique ensuite ces propriétés au texte sélectionné.

Fermez la fenêtre `Visual Basic Editor`, en sélectionnant la commande `Fermer et retourner à Microsoft Excel` du menu `Fichier`.

Améliorer la macro

Sélectionnez maintenant une cellule dont la police et le corps sont différents de ceux de la plage sélectionnée lors de l'enregistrement de la macro. Tapez le raccourci clavier affecté à la macro (`Ctrl+B`). Celle-ci s'exécute.

À la Figure 2.9, on constate que les attributs `Gras` et `Italique` ont bien été appliqués, mais la police et le corps du texte ont changé. Tous les arguments en cours dans la boîte de dialogue `Police` ont en effet été pris en compte lors de l'enregistrement de la macro.

Cela apparaît clairement dans la fenêtre de code (voir Figure 2.8) : les arguments `.Size = 10` et `.Name = "Arial"` du texte de la macro correspondent à la police et au corps du texte sélectionné lors de l'enregistrement de la macro.

Figure 2.9

L'ensemble des arguments en cours lors de l'enregistrement de la macro est appliqué.

	Valérie Marie	Hervé Dubeuf	Hélène Legrand	Ludovic Bidault	Noël CAPLIER	Mathias Fried	Marie Duchêne	Joël Martin
1	75	10	09	18	01	04	02	45
2	92	21	12	22	03	05	08	77
3	95	25	16	28	07	06	14	94
4		39	17	29	15	11	27	78
5		51	19	35	26	13	59	91
6		52	23	36	38	30	60	
7		54	24	37	42	34	62	
8		55	31	41	43	48	76	
9		57	32	44	63	66	80	
10		58	33	49	69	83	93	
11		67	40	50	73	84		
12		68	46	53	74	98		

Pour remédier à ce problème, supprimez les attributs indésirables directement à partir de la fenêtre de code de la macro :

1. Activez l'onglet `Développeur` du ruban, puis cliquez sur le bouton `Macros` du groupe `Code`. Sélectionnez `GrasItalique`, puis cliquez sur le bouton `Modifier`. `Visual Basic Editor` s'ouvre sur la fenêtre de code de la macro `GrasItalique`.

- Supprimez toutes les propriétés de l'objet Font que la macro ne doit pas modifier (toutes les instructions sauf `.FontStyle = "GrasItalique"`). Dans le menu Fichier, choisissez Enregistrer PERSONAL.XLSB ou cliquez sur le bouton Enregistrer de la barre d'outils Standard.

Le texte de la macro doit se présenter ainsi :

```
Sub GrasItalique()
    With Selection.Font
        .FontStyle = "Gras italique"
    End With
End Sub
```

Choisissez Fichier > Fermer. Le tour est joué.



La structure With...End With est utilisée pour paramétrer les propriétés d'un objet sans avoir à répéter la référence à cet objet pour chaque propriété. Puisque la macro ne définit ici qu'une propriété, il est inutile d'utiliser cette structure. La macro se présente alors ainsi :

```
Sub GrasItalique()
    Selection.Font.FontStyle = "Gras italique"
End Sub
```

Au fur et à mesure que vous avancerez dans l'apprentissage de la programmation Excel, vous découvrirez par la pratique les différents éléments des boîtes de dialogue Macro et Enregistrer une macro. Le Tableau 2.2 en présente rapidement les fonctions.

Tableau 2.2 : Fonctions des boîtes de dialogue Macro et Enregistrer une macro

<i>Bouton</i>	<i>Description</i>
Boîte de dialogue Macro	
Exécuter	Exécute la macro sélectionnée – dont le nom apparaît dans la zone de texte Nom de la macro.
Annuler	Ferme la boîte de dialogue Macro.
Pas à pas détaillé	Ouvre la fenêtre de code de la macro sélectionnée dans Visual Basic Editor et l'exécute étape par étape (instruction par instruction). Cette commande constitue un précieux outil de débogage pour vos macros.
Modifier	Ouvre la fenêtre de code de la macro sélectionnée dans Visual Basic Editor afin d'en permettre la modification.

<i>Bouton</i>	<i>Description</i>
Créer	Ouvre, dans Visual Basic Editor, une fenêtre Code simplement composée des instructions <code>Sub NomMacro()</code> et <code>End Sub</code> . Pour accéder à ce bouton, il faut auparavant saisir un nom de macro dans la zone Nom de la macro. Ce nom ne peut être le même que celui d’une macro existante.
Supprimer	Supprime la macro sélectionnée. Un message vous demandant de confirmer la suppression de la macro s’affiche.
Options	Ouvre la boîte de dialogue Options de macro pour la macro sélectionnée, permettant de lui attribuer un raccourci clavier et d’en modifier la description.
Zone de texte Nom de la macro	Permet de désigner une macro existante ou de saisir le nom d’une nouvelle macro. Lorsque vous sélectionnez une macro dans la liste des macros disponibles, son nom s’affiche dans cette zone de texte.
Zone de texte Macros dans	Permet de désigner le classeur dont vous souhaitez afficher les macros ¹ .
Boîte de dialogue Enregistrer une macro	
Nom de la macro	Nom de la macro qui sera enregistrée. Si le nom spécifié est déjà attribué à une macro existante, l’application hôte affichera une boîte de dialogue vous demandant de confirmer le remplacement de la macro.
Touche de raccourci	Permet d’affecter un raccourci clavier à la macro que l’on souhaite enregistrer.
Enregistrer la macro dans	Désigne le classeur où sera stockée la macro ¹ . Le lieu de stockage d’une macro détermine à partir de quels documents la macro sera disponible, c’est-à-dire où elle pourra être exécutée, modifiée ou supprimée.
Description	Destinée à la saisie d’une description de la macro. Par défaut, la date de création et le créateur apparaissent dans cette zone.
Bouton OK	Démarre l’enregistrement de la macro sans qu’aucun raccourci ne lui soit attribué.
Bouton Annuler	Ferme la boîte de dialogue Enregistrer une macro sans déclencher l’Enregistreur de macro.

1. Le stockage et la disponibilité des macros sont traités à la fin de ce chapitre.

Une autre méthode d’enregistrement

L’Enregistreur de macro est un instrument souple qui enregistre l’ensemble des commandes que vous exécutez dans l’application hôte. Vous pouvez donc enregistrer une macro en utilisant n’importe laquelle des méthodes que propose l’application hôte pour exécuter les commandes que vous souhaitez intégrer à la macro.

Dans le cas de la macro *GrasItalique*, il est plus simple de cliquer successivement sur les icônes Gras et Italique de la barre d'outils que de passer par la boîte de dialogue Format de cellule. Rien ne vous empêche d'enregistrer votre macro de la même façon.

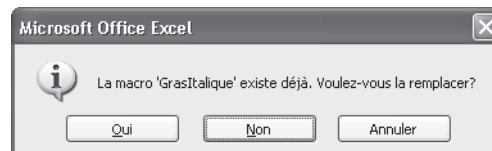
Enregistrement

Pour réenregistrer la macro *GrasItalique* :

1. Sélectionnez une cellule.
2. Cliquez sur le bouton Enregistrer une macro de l'onglet Développeur. Dans la zone Nom de la macro de la boîte de dialogue Enregistrer une macro, saisissez *GrasItalique*.
3. Affectez un raccourci clavier à la macro et saisissez une brève description dans la zone Description
4. Cliquez sur le bouton OK. Une boîte de dialogue s'affiche, vous demandant de confirmer le remplacement de la macro existante. Confirmez.

Figure 2.10

Confirmez le remplacement de la macro GrasItalique.



5. Cliquez tour à tour sur les icônes Gras et Italique de la barre d'outils.
6. Cliquez sur le bouton Arrêter l'enregistrement de la barre d'outils Arrêt de l'enregistrement.

La macro est enregistrée.

Vous pouvez aussi enregistrer la macro en utilisant les raccourcis clavier affectés aux enrichissements Gras et Italique – respectivement Ctrl+G et Ctrl+I.

Structure de la macro

Observons la façon dont ces actions ont été codées en Visual Basic. Ouvrez la Fenêtre de code de la macro *GrasItalique* (bouton Macro de l'onglet Développeur, puis Modifier).

Le texte de la macro se présente ainsi :

```
Sub GrasItalique()  
    Selection.Font.Bold = True  
    Selection.Font.Italic = True  
End Sub
```

Les propriétés **Bold** et **Italic** de l'objet **Font** sont définies à **True**, indiquant que les cellules sélectionnées seront enrichies des attributs gras et italique.

Remarquez l'absence de la structure **With...End With**. Cette structure n'est utilisée que lorsque plusieurs propriétés d'un objet sont validées dans une seule action – c'est le cas pour toutes les options d'une boîte de dialogue au moment où vous cliquez sur le bouton **OK**.

Vous pouvez cependant utiliser cette structure afin d'améliorer la lisibilité de la macro. Elle doit alors se présenter ainsi :

```
Sub GrasItalique()  
    With Selection.Font  
        .Bold = True  
        .Italic = True  
    End With  
End Sub
```

Écrire la macro

Maintenant que vous connaissez la structure de la fenêtre de code d'une macro, vous allez écrire directement la macro, sans l'aide de votre programmeur attitré, l'Enregistreur de macro.

Pour écrire la macro **GrasItalique** :

1. Cliquez sur le bouton **Macros** de l'onglet **Développeur**. Sélectionnez la macro **GrasItalique** dont le nom s'affiche alors dans la zone **Nom** de la macro. Cliquez sur le bouton **Supprimer**. Excel vous demande de confirmer la suppression de la macro **GrasItalique**. Confirmez.
2. Cliquez de nouveau sur le bouton **Macros** de l'onglet **Développeur**. Dans la zone **Nom** de la macro, saisissez **GrasItalique**, puis cliquez sur le bouton **Créer**. **Visual Basic Editor** s'ouvre sur la fenêtre de code de la nouvelle macro **GrasItalique**.

Le texte de la macro se présente sous sa forme minimale :

```
Sub GrasItalique()  
End Sub
```

3. Insérez une ligne entre **Sub GrasItalique()** et **End Sub**. Saisissez simplement le texte de la macro tel que nous l'avons vu lors de la section précédente.
4. Dans le menu **Fichier**, choisissez **Enregistrer PERSONAL.XLSB**, puis **Fermer** et retourner dans **Microsoft Excel**.

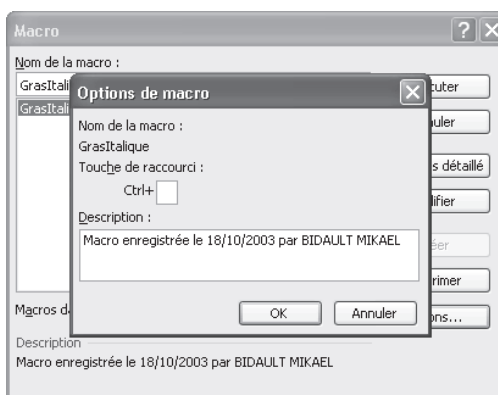
Créer une macro n'est pas plus compliqué que cela.

Contrairement à la méthode de l'enregistrement, la *création* d'une macro ne permet pas l'attribution d'un raccourci clavier. Pour affecter un raccourci clavier à la macro GrasItalique, procédez comme suit :

1. Choisissez Outils > Macro > Macros ou, si vous utilisez Excel 2007, cliquez sur le bouton Macros de l'onglet Développeur.
2. Sélectionnez la macro GrasItalique, puis cliquez sur le bouton Options.
3. Dans la boîte de dialogue Options de macro qui s'affiche, indiquez un raccourci clavier et saisissez éventuellement une description pour la macro (voir Figure 2.11). Validez en cliquant sur OK

Figure 2.11

La boîte de dialogue Options de macro permet d'affecter un raccourci clavier à une macro existante.



Info

Visual Basic pour Applications ne tient pas compte des majuscules. Celles-ci sont placées dans le code (`Selection.Font`, par exemple) dans le seul but d'en faciliter la lecture. Mais vous pouvez parfaitement saisir du texte dans une fenêtre de code entièrement en minuscules (`selection.font`).

Astuce

Si vous saisissez du texte en minuscules dans une fenêtre de code, Visual Basic remplace les majuscules dans les instructions qu'il reconnaît lorsque vous changez de ligne. S'il ne modifie pas la casse d'une instruction saisie en minuscules, c'est qu'il ne la reconnaît pas. Par exemple, `selection.font.bold = true` deviendra `Selection.Font.Bold = True` lors du changement de ligne ; par contre, si vous tapez `selection.font.old = true`, Word ne placera pas de capitale à `old`. C'est un bon moyen de vérifier que vous n'avez pas commis de fautes lors de la saisie.

Exécution de la macro

Une macro *créée* s'exécute exactement de la même façon qu'une macro *enregistrée*. Vous pouvez exécuter la macro GrasItalique, soit à partir de la boîte de dialogue Macros, soit en utilisant le raccourci clavier que vous lui aurez attribué après.

Vous savez maintenant enregistrer (selon la méthode de votre choix) et créer une macro. Si la macro GrasItalique vous paraît anodine, sachez que les principes acquis dans les sections précédentes sont valables pour toutes les macros, quelle que soit l'application hôte.

La macro GrasItalique est une véritable commande que vous avez ajoutée à Excel. En procédant de la même façon, vous pouvez créer n'importe quelle commande, en fonction de vos besoins : insérer un titre WordArt et lui appliquer une mise en forme spécifique, définir les valeurs de cellules, etc.

Info

Vous avez appris dans ce chapitre à enregistrer ou à créer une macro. La mise en œuvre de macros complexes nécessite souvent de combiner ces deux méthodes. On enregistre en général les commandes de la macro, puis on y écrit les fonctions qui ne peuvent être enregistrées.

Astuce

Pour qu'une macro s'exécute automatiquement à l'ouverture d'un classeur, affectez-lui le nom Auto_Open. Cette fonction est intéressante si vous souhaitez paramétrer différemment Excel selon les classeurs affichés. Enregistrez simplement les options d'Excel dans une macro Auto_Open.

Pour qu'une macro s'exécute automatiquement à la fermeture d'un classeur, affectez-lui le nom Auto_Close. Vous pouvez ainsi mettre à jour un autre fichier, créer une sauvegarde du fichier dans un autre dossier, etc.

Choisir l'accessibilité des macros

Lorsque vous enregistrez ou créez des macros, celles-ci sont stockées dans un *projet* VBA, attaché à un document spécifique de l'application hôte. Pour exécuter une macro, le document hébergeant le projet de stockage de la macro doit être actif. Le stockage des macros est donc une donnée fondamentale, puisqu'il en détermine l'accessibilité pour l'utilisateur final. Une bonne gestion des macros est le préalable à une application puissante et efficace.

Accessibilité globale ou limitée

Une macro peut être accessible – c'est-à-dire exécutée, modifiée, renommée ou supprimée – à partir de n'importe quel document, ou limitée à des documents spécifiques. S'il est

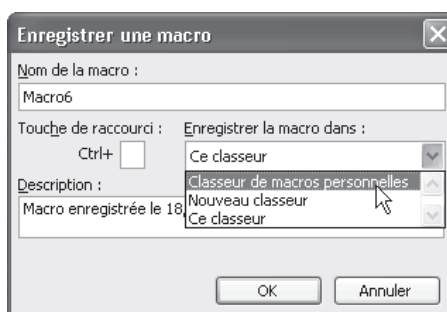
intéressant de pouvoir assurer une accessibilité globale aux macros, il est parfois préférable d'attacher une macro à un classeur spécifique. C'est le cas si la macro est conçue pour fonctionner avec un certain type de données et si elle est inutile dans d'autres classeurs – voire susceptible d'y provoquer des dommages. En outre, limiter la disponibilité des macros aux documents concernés en facilite la gestion.

Le stockage d'une macro est défini lors de son enregistrement ou de sa création, et peut être modifié par la suite. Une macro peut également être créée directement à partir de Visual Basic Editor, sans passer par la boîte de dialogue Macros ni par la boîte de dialogue Enregistrer une macro. Le stockage de la macro est alors déterminé dans l'Explorateur de projet. L'Explorateur de projet et la création de macros à partir de Visual Basic Editor sont respectivement présentés aux Chapitres 4 et 5.

Lors de l'enregistrement d'une macro, l'affectation de la macro à un document s'effectue par la zone de liste Enregistrer la macro dans de la boîte de dialogue Enregistrer une macro (voir Figure 2.12). Les sections suivantes présentent les possibilités de stockage des macros Excel.

Figure 2.12

Sélectionnez le document de stockage de la macro lors de son enregistrement.



Classeurs et modèles

Les macros enregistrées dans Excel sont stockées dans des classeurs ou dans des modèles. Pour accéder à une macro, il faut que le classeur dans lequel est stockée la macro soit ouvert. Si plusieurs classeurs sont ouverts, vous pouvez accéder aux macros d'un des classeurs ouverts à partir de n'importe quel autre classeur.

Les macros enregistrées dans un modèle sont accessibles lorsque vous créez un nouveau classeur fondé sur ce modèle (en choisissant la commande Nouveau de l'onglet Fichier et en sélectionnant un modèle). Lorsque vous enregistrez le nouveau classeur, les macros du modèle sont "copiées" dans celui-ci, et restent donc disponibles par la suite, lorsque vous rouvrez le classeur.

Notez cependant que les classeurs Excel n'entretiennent pas de lien avec le modèle à partir duquel ils ont été créés. Si vous ajoutez, modifiez ou supprimez des macros dans un modèle, ces changements ne seront pas effectifs pour les classeurs préalablement créés à partir du modèle.

Pour enregistrer ou créer une macro dans un modèle, vous devez ouvrir le modèle en question.

Les classeurs dans lesquels sont stockées les macros sont identifiés par une extension spécifique. Il peut s'agir d'un "modèle prenant en charge les macros" (.xlsm) ou d'un "classeur prenant en charge les macros" (.xslm).



Dans les versions antérieures à Excel 2007, les classeurs prenant en charge les macros ne portent pas d'extension spécifique.

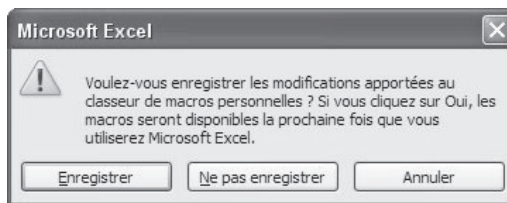
Le classeur de macros personnel

Lors de l'enregistrement de vos macros, vous pouvez choisir de stocker la macro dans le classeur actif afin d'en limiter la disponibilité à ce dernier. La macro ne pourra alors être exécutée, modifiée ou supprimée qu'à condition que ce classeur soit ouvert. Mais vous aurez certainement besoin d'accéder à la plupart de vos macros à partir de classeurs différents. Pour rendre une macro accessible à partir de n'importe quel classeur Excel, il suffit de l'enregistrer dans le classeur de macros personnel, PERSONAL.XLSB.

Le classeur PERSONAL.XLSB est créé la première fois que vous enregistrez une macro dans le classeur de macros personnel. Lorsque vous quittez Excel après avoir enregistré votre première macro dans le classeur de macros personnel, la boîte de dialogue présentée à la Figure 2.13 s'affiche.

Figure 2.13

Le classeur de macros personnel est créé lorsque vous quittez Excel après avoir enregistré ou écrit votre première macro dans le classeur de macros personnel.



Le classeur de macros personnel est ouvert chaque fois que vous exécutez Excel. Vous pouvez donc en exécuter les macros qui y sont stockées à partir de n'importe quel classeur.



Par défaut, le classeur de macros personnel est masqué au lancement d'Excel. Pour accéder à ce classeur, choisissez la commande Afficher de l'onglet Affichage, et sélectionnez PERSONAL.XLSB dans la boîte de dialogue Afficher.



Le classeur de macros personnel est stocké dans le dossier XLSTART. Ce classeur contiendra probablement l'essentiel de vos macros. Il est donc conseillé d'en effectuer régulièrement une sauvegarde. L'emplacement de ce fichier varie en fonction de votre système d'exploitation, et aussi d'une version d'Excel à l'autre.

Pour en connaître l'emplacement, afficher le fichier PERSONAL.XLSB et choisissez la commande Enregistrer sous. Dans la boîte de dialogue qui apparaît, relevez le chemin proposé par défaut.

Les macros complémentaires

Les macros peuvent également être attachées à un classeur enregistré en tant que macro complémentaire (extension XLAM et XLA pour les versions antérieures à Excel 2007). Ce type de classeur est particulièrement adapté à la distribution de macros. Les classeurs de macros complémentaires peuvent en effet être "chargés" dans Excel. Les macros qui y sont contenues sont alors rendues accessibles au lancement de l'application. Contrairement au classeur de macros personnel, les classeurs de macros complémentaires ne sont pas "ouverts" : les macros qu'ils contiennent sont chargées en mémoire et les barres d'outils et menus d'Excel sont mis à jour pour intégrer les fonctionnalités qu'apportent les macros.

L'enregistrement au format Macro complémentaire est indéniablement la solution adaptée si vous développez des solutions complètes par modules. Vous évitez ainsi de surcharger le classeur de macros personnel et pouvez regrouper les macros par classeur tout en leur assurant une accessibilité globale. Les principaux avantages des macros complémentaires sont les suivants :

- distribution et gestion simplifiées ;
- possibilité d'activation/désactivation très simple ;
- économie de ressources mémoire ;
- exécution plus rapide des macros ;
- ajout de commandes à l'application de façon transparente pour l'utilisateur ;
- les macros complémentaires chargées n'apparaissent pas dans la liste des macros.

Enregistrer une macro complémentaire

Pour enregistrer un classeur en tant que macro complémentaire, commencez par préparer le projet VBA lui-même. Accédez à Visual Basic Editor :

1. Commencez par vous assurer que vos programmes VBA fonctionnent correctement et ne contiennent pas de bogues.
2. Dans Visual Basic Editor, ouvrez un module du projet et choisissez Débogage > Compiler.
3. Protégez éventuellement votre projet par mot de passe (voir Chapitre 16).
4. Quittez ensuite Visual Basic Editor et, dans Excel affichez les Propriétés du fichier en cliquant sur l'onglet Fichier, puis sur Informations. Dans le volet droit de la fenêtre qui s'affiche, cliquez sur le bouton Propriétés, puis choisissez Propriétés avancées.



Avec Excel 2007, cliquez sur le bouton Office, puis choisissez Préparer > Propriétés. Avec les versions antérieures d'Excel, choisissez la commande Propriétés du menu Fichier.

5. Activez l'onglet Résumé de la boîte de dialogue Propriétés. Saisissez un nom et un descriptif représentatifs dans les zones Titre et Commentaire (voir Figure 2.14). Ce sont le titre et le commentaire qui apparaîtront dans la boîte de dialogue Macros complémentaires.

Figure 2.14

Choisissez un titre et un commentaire clairs.

The screenshot shows a dialog box titled 'Propriétés de test.xls' with a close button (X) in the top right corner. It has five tabs: 'Général', 'Résumé', 'Statistiques', 'Contenu', and 'Personnalisation'. The 'Résumé' tab is selected. The fields are as follows: 'Titre' contains 'Outils d'analys FGH'; 'Sujet' is empty; 'Auteur' contains 'BIDAULT MIKAEL'; 'Responsable' is empty; 'Société' contains 'aucune' with a mouse cursor pointing at it; 'Catégorie' is empty; 'Mots clés' is empty; 'Commentaires' contains 'Ensemble d'outils d'analyse financière destinés à la société FGH (accessibles via le menu Outils FGH)'; 'Répertoire Web' is empty; 'Modèle' is empty. At the bottom, there is a checkbox labeled 'Enregistrer l'image de l'aperçu' which is unchecked. At the very bottom are 'OK' and 'Annuler' buttons.

6. Choisissez Fichier > Enregistrer sous. Dans la zone Type de fichier, sélectionnez Macro complémentaire Excel (*.xlam). Le dossier *AddIns* est activé par défaut.



Excel 2010 et Excel 2007 proposent deux formats pour les macros complémentaires : le format .xlam et le format Excel 97-2003 (extension .xla). Si vous souhaitez distribuer vos macros, il est recommandé de choisir ce second format, afin d'assurer la compatibilité de vos macros avec un maximum de versions d'Excel.

7. Affectez un nom adapté et cliquez sur Enregistrer.

La macro complémentaire est enregistrée et automatiquement fermée.

Activer/désactiver une macro complémentaire

Pour activer ou désactiver une macro complémentaire, procédez comme suit :

1. Cliquez sur l'onglet Fichier, puis sur le bouton Options. Dans la fenêtre qui s'affiche, sélectionnez Compléments dans le volet gauche. La fenêtre représentée à la Figure 2.15 s'affiche. Cliquez sur le bouton Atteindre à côté de la zone Gérer.

Les macros complémentaires sont affichées. Les macros complémentaires actives sont cochées.



Avec Excel 2007, cliquez sur le Bouton Office au lieu de l'onglet Fichier. Dans les versions antérieures, choisissez Outils > Macros complémentaires ou Outils > Compléments.

2. Cliquez sur le bouton Parcourir. Si vous avez enregistré la macro complémentaire dans le dossier proposé par défaut, elle apparaît dans la liste. Sélectionnez le fichier de macro complémentaire voulu et validez.
3. Cochez ou décochez la macro complémentaire. Le nom de la macro complémentaire et le descriptif qui apparaissent dans la boîte de dialogue Macros complémentaires sont ceux qui ont été indiqués comme titre et commentaire du fichier (voir Figure 2.16).

Figure 2.15
La liste des compléments d'Excel 2007.

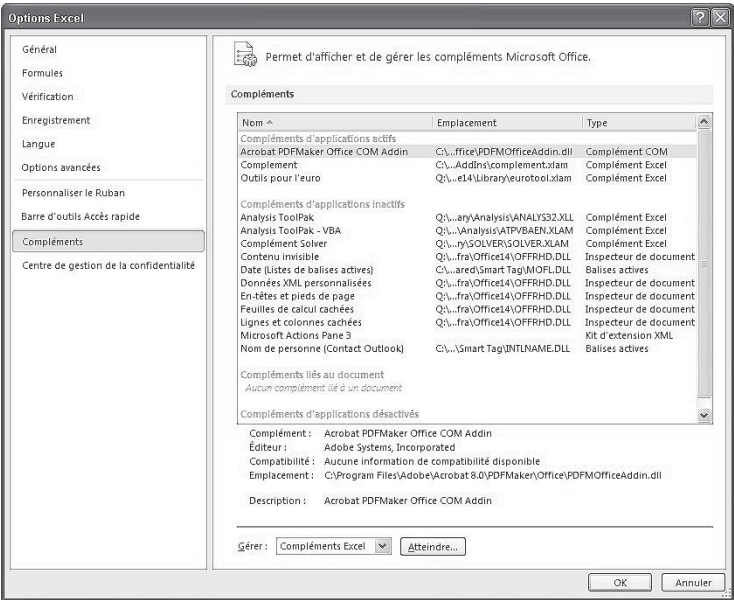


Figure 2.16
Notre macro complémentaire apparaît maintenant dans la liste des macros complémentaires d'Excel.



Les macros complémentaires intégrées d'Excel

Excel est livré avec un nombre important de macros complémentaires, telles que les Outils pour l'euro ou le Solveur, etc. Celles-ci doivent être installées pour être accessibles. Si ce n'est pas le cas, vous devez relancer l'installation d'Office et installer les macros complémentaires non disponibles.



Avant de vous lancer dans des activités de programmation complexe, vérifiez s'il n'existe pas une macro complémentaire intégrée répondant à vos besoins. Vous pouvez également télécharger gratuitement des macros complémentaires sur le site de Microsoft (<http://www.microsoft.com/france>). Enfin, des sociétés de développement spécialisées commercialisent des macros complémentaires.

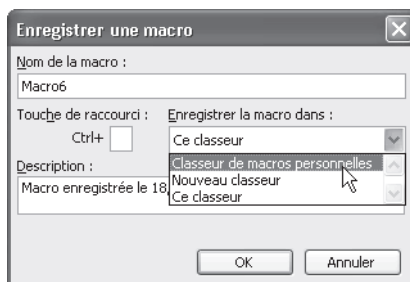
Définir le classeur de stockage lors de l'enregistrement d'une macro

Lors de l'enregistrement d'une macro, la zone Enregistrer la macro dans de la boîte de dialogue Enregistrer une macro, permet de définir le classeur dans lequel sera stockée la macro. Trois options sont disponibles :

- **Classeur de macros personnel.** La macro sera enregistrée dans le classeur de macros personnel, et accessible à partir de n'importe quel classeur.
- **Nouveau classeur.** Un nouveau classeur est créé et la macro y est stockée.
- **Ce classeur.** La macro est stockée dans le classeur actif. Il peut s'agir du classeur de macros personnel (PERSONAL.XLSB) ou d'un autre classeur.

Figure 2.17

Définissez le classeur qui hébergera une macro lors de l'enregistrement de cette dernière.



Accéder aux macros d'un classeur spécifique

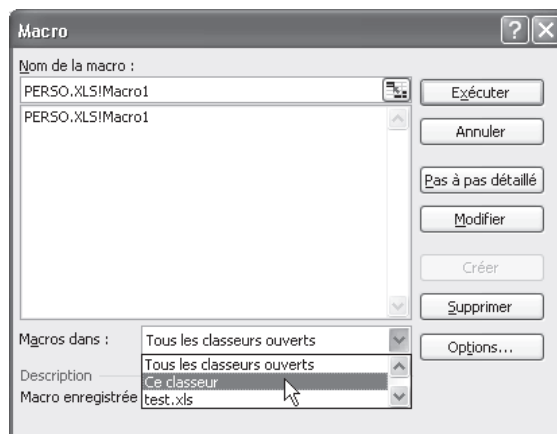
La zone de liste déroulante Macros de la boîte de dialogue Macro permet de définir les macros que vous souhaitez visualiser. Quatre options s'offrent à vous :

- **Tous les classeurs ouverts.** Les macros de tous les classeurs ouverts (y compris le classeur de macros PERSONAL.XLSB) sont accessibles. Les macros qui ne sont pas stockées dans le classeur de macros personnel apparaissent sous la forme *Classeur!NomMacro*, où *Classeur* est le nom du classeur hébergeant la macro, et *NomMacro* le nom de la macro.

- **Ce classeur.** Seules les macros du classeur actif sont affichées. Les macros apparaissent alors simplement sous la forme NomMacro.
- **PERSONAL.XLSB.** Seules les macros du classeur de macros personnel sont accessibles.
- **Nom_Classeur.** Vous pouvez aussi choisir de ne visualiser que les macros de l'un des classeurs ouverts, en sélectionnant simplement son nom dans la liste.

Figure 2.18

Vous pouvez définir le classeur dont vous souhaitez visualiser les macros.

**Attention**

Si vous fermez le classeur de macros personnel, vous ne pourrez plus accéder aux macros qui y sont stockées, ni y enregistrer de nouvelles macros. Le classeur de macros personnel s'ouvrira de nouveau lors de la prochaine session Excel. Pour enregistrer des macros d'accès global ou accéder aux macros du classeur de macros personnel au cours de la session active, vous devez rouvrir le classeur PERSONAL.XLSB.

Astuce

Le classeur PERSONAL.XLSB s'ouvre à l'exécution d'Excel parce qu'il se trouve dans le dossier XLSTART (ou XLOuvrir selon la version d'Office). Pour ouvrir automatiquement un classeur au lancement d'Excel, créez un raccourci vers ce classeur et placez-le dans ce dossier.



3

Déplacement et sélection dans une macro Excel

Au sommaire de ce chapitre

- Méthodes de sélection dans une feuille Excel
- Coder les déplacements effectués lors de l'enregistrement d'une macro

Le déplacement et la sélection de cellules dans un classeur constituent une donnée primordiale lors de l'enregistrement de macros. Il est indispensable de connaître les techniques de déplacement dans une feuille de calcul, et de comprendre les concepts de référence relative ou absolue aux cellules, pour créer des macros qui se comportent comme vous le souhaitez.

Pour enregistrer des déplacements dans une feuille Excel, vous pouvez utiliser indifféremment le clavier ou la souris. L'emplacement de la cellule active est enregistré lorsque vous effectuez une opération (mise en forme, saisie, etc.) qui modifie la feuille Excel. Autrement dit, si vous vous contentez de vous déplacer dans la feuille – par des clics de souris ou en utilisant les touches de déplacement du clavier – sans jamais intervenir sur le contenu ou la mise en forme de la cellule, ces déplacements ne seront pas enregistrés dans la macro, puisqu'ils ne modifient en rien la feuille.

La sélection d'éléments dans une feuille Excel repose sur l'objet Range qui représente une cellule, une ligne, une colonne ou une combinaison de ces éléments. Comme vous le verrez dans cette section, les propriétés utilisées varient selon le type de sélection effectué, mais toutes renvoient un objet Range.

Le codage en langage Visual Basic de vos déplacements dans la feuille de calcul Excel varie selon que vous activez ou non la référence relative aux cellules, en cliquant sur le bouton correspondant de la barre d'outils Arrêt de l'enregistrement (voir Figure 3.1) ou dans la zone Code de l'onglet Développeur si vous utilisez Excel 2007.



Figure 3.1

Vous pouvez enregistrer vos déplacements par référence relative ou absolue aux cellules.

Méthodes de sélection dans une feuille Excel

Clavier

L'enregistrement de déplacements dans une feuille Excel, par référence relative aux cellules, nécessite, dans certains cas, que vous utilisiez le clavier (par exemple, si vous souhaitez activer la dernière cellule non vide d'une ligne). Le Tableau 3.1 présente les différentes possibilités de déplacement dans Excel à l'aide du clavier.

Tableau 3.1 : Déplacement dans une feuille Excel à l'aide du clavier

<i>Pour se déplacer</i>	<i>Clavier</i>
D'une cellule vers la droite	→
D'une cellule vers la gauche	←
D'une cellule vers le haut	↑
D'une cellule vers le bas	↓
Au début de la ligne courante	Début
Sur la première cellule non vide de la ligne courante ¹	Ctrl+←
Sur la dernière cellule non vide de la ligne courante ¹	Ctrl+→
Sur la première cellule non vide de la colonne courante ¹	Ctrl+↑
Sur la dernière cellule non vide de la colonne courante ¹	Ctrl+↓
Sur la cellule située à l'angle supérieur gauche de la feuille active (A1)	Ctrl+Début
Sur la cellule située à l'angle inférieur droit de la feuille active ²	Ctrl+Fin

1. Si des cellules vides se trouvent entre la cellule active et la cellule visée, Excel s'arrête successivement sur les cellules contiguës aux cellules vides. Par exemple, si la cellule B10 est active et si toutes les cellules situées au-dessus d'elle contiennent des données à l'exception de la cellule B5, la répétition de la combinaison de touches Ctrl+↑ entraînera l'activation successive des cellules B6, puis B4 et enfin B1.
2. L'adresse de la cellule située à l'angle inférieur droit de la feuille active est la combinaison de la dernière colonne de la dernière ligne contenant des données. Cette cellule peut être vide.



Lors de l'enregistrement d'une macro par référence absolue aux cellules, c'est l'adresse des cellules qui est enregistrée. Pour enregistrer un déplacement relatif (par exemple, la dernière cellule non vide de la ligne courante), vous devez activer l'enregistrement par référence relative aux cellules en cliquant sur le bouton correspondant de la barre d'outils Arrêt de l'enregistrement.

Pour étendre la sélection de la cellule active à une cellule donnée de la feuille, utilisez l'une des combinaisons de touches présentées dans le Tableau 3.1, en maintenant la touche Maj enfoncée. Pour sélectionner les colonnes entières correspondant aux cellules sélectionnées, utilisez le raccourci clavier Ctrl+Barre d'espace ; pour sélectionner les lignes entières, utilisez Maj+Barre d'espace.

Notez que, pour sélectionner des zones non contiguës, vous devez obligatoirement utiliser la souris.

Souris

L'utilisation de la souris pour effectuer des sélections dans Excel est très simple. Pour sélectionner une cellule, cliquez dessus. Pour sélectionner une ligne ou une colonne, cliquez sur l'en-tête de la ligne ou de la colonne concernée.

Pour sélectionner des cellules adjacentes, cliquez sur la première cellule de la plage que vous souhaitez sélectionner, appuyez sur la touche Maj et, tout en la maintenant enfoncée, cliquez sur la dernière cellule de la plage. Pour sélectionner des lignes ou des colonnes adjacentes, procédez de la même façon, en cliquant sur les références de ces lignes ou de ces colonnes.

Pour sélectionner des cellules non contiguës, cliquez sur la première cellule que vous souhaitez sélectionner, appuyez sur la touche Ctrl et, tout en la maintenant enfoncée, cliquez successivement sur les cellules que vous souhaitez sélectionner. Pour sélectionner des lignes ou des colonnes non contiguës, procédez de la même façon, en cliquant sur les références de ces lignes ou de ces colonnes.

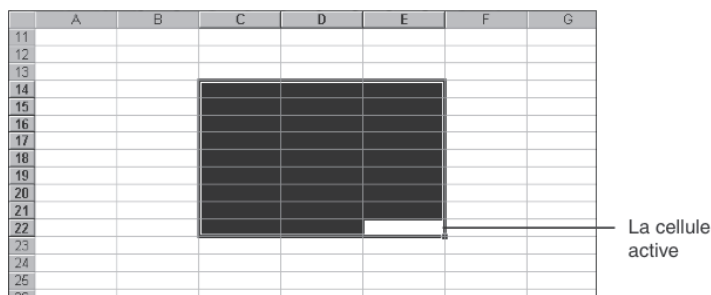
Vous pouvez combiner la sélection d'éléments non contigus de la feuille en maintenant la touche Ctrl enfoncée et en cliquant sur les éléments voulus. Vous pouvez, par exemple, sélectionner simultanément la colonne C, la ligne 5 et la cellule F4. Cliquez sur l'en-tête de la colonne C, appuyez sur la touche Ctrl et, tout en la maintenant enfoncée, cliquez sur l'en-tête de la ligne 5, puis sur la cellule F4.

Notion de cellule active

Lorsqu'une plage de cellules est sélectionnée dans une feuille Excel, toutes les cellules de cette plage, à l'exception d'une seule, sont noircies. La cellule non noircie est la cellule active de la plage (voir Figure 3.2).

Figure 3.2

Dans une plage de cellules Excel, une seule cellule est la cellule active.



Si vous appliquez une mise en forme (une police particulière, par exemple), cette mise en forme sera appliquée à l'ensemble des cellules de la plage sélectionnée. Si vous appuyez

sur la touche Suppr, le contenu de toutes les cellules de la plage sera supprimé. Plus généralement, si vous effectuez une opération pouvant affecter simultanément plusieurs cellules, elle s'appliquera à l'ensemble des cellules de la plage sélectionnée.

Cependant, certaines opérations – comme la saisie de texte ou de formules – ne peuvent s'appliquer qu'à une cellule à la fois. C'est alors la cellule active de la plage sélectionnée qui est affectée. Par exemple, si la plage de cellules A5+F10 est sélectionnée, et si la cellule active est la cellule A5, le texte saisi au clavier sera inséré dans la cellule A5, et la plage A5+F10 restera sélectionnée.



Pour nommer une plage de cellules, on indique la cellule située à l'angle supérieur gauche de la plage (A5), puis la cellule située à l'angle inférieur droit de la plage (F10).

La cellule active d'une plage de cellules dépend de l'ordre dans lequel vous avez sélectionné les différentes cellules qui la composent. Vous verrez dans les sections qui suivent comment le mode de sélection détermine la cellule active et comment la cellule active est codée en langage Visual Basic.

Références relatives et références absolues

Par défaut, l'enregistrement s'effectue par référence absolue à des cellules. Cela signifie que, lorsque vous vous déplacez dans les cellules d'une feuille Excel, l'Enregistreur de macro mémorise l'adresse de la cellule (combinaison du numéro de ligne et de la lettre de colonne). Ainsi, si vous enregistrez un déplacement sur la cellule B6, l'exécution de la macro entraînera l'activation de la cellule B6, et ce, quelle que soit la cellule active au moment de l'exécution de la macro.

Le bouton Référence relative de la barre d'outils Arrêt de l'enregistrement permet d'enregistrer les déplacements dans la feuille Excel relativement à la cellule initialement active. Ce n'est plus l'adresse de la cellule qui est prise en considération, mais le déplacement dans la feuille. Ainsi, un déplacement de la cellule B5 à la cellule C7 sera enregistré comme un déplacement d'une colonne vers la droite et de deux lignes vers le bas. Si, au moment de l'exécution de la macro, la cellule active est la cellule D1, la macro entraînera l'activation de la cellule E3 (située une colonne à droite et deux lignes en dessous de la cellule D1).

Lors de l'enregistrement de macros dans Excel, vous pouvez combiner les références relatives et absolues, en cliquant sur le bouton Référence relative chaque fois que vous voulez changer le type de référence aux cellules. Observez le classeur représenté à la Figure 3.3. La colonne D contient les chiffres d'affaires effectués par les représentants. La colonne E doit contenir les primes.

Figure 3.3

La combinaison des références relatives et absolues permettra de calculer les primes des représentants.

	A	B	C	D	E
1	Représentant	Région	Ref	Chiffre du mois	Prime
4	Artuis J.	Ouest	0078	54 255	
5	Bertrand Isabelle	Nord-est	0072	89 650	
6	Bidault Jean	Est	0079	48 625	
7	Boitier Henri	Ouest	0008	78 545	
8	Cartois Hervé	Ouest	0077	102 565	
9	Denis Emmanuelle	Sud	0074	35 007	
10	Dupin Stéphane	Sud-ouest	0073	69 854	
11	Frimousse Sylvie	Sud-est	0098	75 215	
12	Goraguer Vivienne	Centre	0001	59 303	
13	Groppois Yves	Nord	0097	82 350	
14	Jean Mickael	Centre	0128	49 950	
15	Le Bras Laetitia	Centre	0178	28 500	
16	Lemaire Francis	Nord-Ouest	0139	95 860	
17	Les Neury Yvonne	Ouest	0005	45 230	
18	Letendre Alain	Est	0145	98 500	
19	Limouse Evelyne	Sud-est	0153	68 500	

Pour calculer la prime, il suffit de procéder comme suit :

1. Se placer dans la première cellule contenant un chiffre d'affaires (D4).
2. Effectuer le calcul de la prime – vous apprendrez au Chapitre 5 à créer des fonctions personnalisées et à les exploiter dans vos programmes VBA.
3. Se déplacer d'une cellule vers la droite (E4).
4. Insérer le résultat issu du calcul.
5. Se déplacer d'une cellule vers le bas, puis d'une cellule vers la gauche, afin d'atteindre la cellule contenant le chiffre suivant (D5).
6. Recommencer au point 2 si la cellule sélectionnée contient une valeur.

Si vous souhaitez créer un programme VBA prenant en charge ce calcul, il vous faudra utiliser une référence absolue aux cellules pour le point 1 et une référence relative aux cellules pour les déplacements des points 3 et 5. Il vous suffit pour cela de vous assurer que l'enregistrement de la macro s'effectue par référence absolue avant d'effectuer la sélection de la cellule A4, puis de cliquer sur le bouton Référence relative, afin d'activer l'enregistrement des déplacements par référence relative aux cellules.



La référence aux cellules (relative ou absolue) active au moment où vous interrompez l'enregistrement d'une macro sera aussi la référence active si vous enregistrez une nouvelle macro dans la même session Excel (sans avoir quitté, puis relancé l'application). Lorsque vous enregistrez une macro, pensez toujours à vérifier que vos déplacements seront enregistrés selon la référence aux cellules souhaitées avant de commencer à vous déplacer dans la feuille Excel.

Coder les déplacements effectués lors de l'enregistrement d'une macro

Les déplacements dans une feuille Excel sont interprétés en Visual Basic comme la manipulation d'objets Excel. Ces objets sont des objets Range auxquels vous accéderez à l'aide des propriétés suivantes :

- **Range.** Renvoie un objet Range représentant une cellule, une plage de cellules, ou un groupe de cellules non contiguës.
- **Cells.** Renvoie la collection Cells qui représente toutes les cellules du classeur actif. Permet aussi de renvoyer une cellule ou une plage de cellules spécifiée.
- **Row.** Renvoie un numéro représentant une ligne de la feuille. La propriété Rows renvoie un objet Range qui représente toutes les lignes de la feuille ou toutes les lignes d'un objet Range spécifié.
- **Column.** Renvoie un numéro représentant une colonne de la feuille. La propriété Columns renvoie un objet Range qui représente toutes les colonnes de la feuille ou toutes les colonnes d'un objet Range spécifié.
- **ActiveCell.** Renvoie un objet Range représentant la cellule active d'une feuille de calcul.
- **Selection.** Renvoie l'objet sélectionné dans la feuille de calcul active. Si l'objet sélectionné est une cellule ou un groupe de cellules, l'objet renvoyé est un objet Range représentant cette cellule ou ce groupe de cellules.
- **Offset.** Renvoie un objet Range qui représente une plage décalée par rapport à la plage spécifiée.

Vous apprendrez à exploiter ces objets à l'aide des méthodes suivantes :

- **Select** et **GoTo.** Sélectionnent l'objet spécifié.
- **Activate.** Active l'objet spécifié. S'il s'agit d'un objet Range, la cellule spécifiée devient la cellule active. Si un groupe de cellules est sélectionné, la sélection est maintenue.
- **Resize.** Modifie l'ampleur d'une plage de cellules.

Référence absolue aux cellules

Cette section présente le codage VBA des déplacements par référence absolue aux cellules.

Sélection de cellules contiguës

L'expression Visual Basic pour une référence absolue à une cellule se présente ainsi :

```
Range("Adresse_Cellule").Select
```

- La propriété Range indique qu'il s'agit d'un objet Range.
- L'argument *Adresse_Cellule* précise l'adresse de cet objet. Cet argument est composé de la référence de colonne (une lettre) immédiatement suivie de la référence de ligne (un nombre).
- La méthode Select entraîne la sélection de l'objet Range précédemment défini (ici la cellule).

Par exemple, l'instruction Visual Basic pour activer la cellule B5 de la feuille active est :

```
Range("B5").Select
```

Pour sélectionner une cellule ou une plage de cellules, la feuille contenant la cellule doit être active. Si tel n'est pas le cas, commencez par activer la feuille voulue à l'aide de la méthode Activate. Les instructions permettant de sélectionner la cellule B5 de la feuille intitulée Janvier du classeur Ventes.xlsx – sans qu'il soit nécessaire que Janvier soit la feuille active – sont :

```
Workbooks("Ventes.xlsx").Sheets("Janvier").Activate  
Range("B5").Select
```



Info

Nous considérerons dans la suite de ce chapitre que la sélection s'effectue sur la feuille active, et omettons donc toute instruction destinée à activer la feuille dont les cellules doivent être sélectionnées.



Astuce

La propriété Cells permet aussi de coder une référence absolue à une cellule. La propriété Cells s'utilise avec la syntaxe suivante :

Cells (ligne, colonne)

où ligne est l'index de ligne et colonne l'index de colonne, tous deux exprimés par un chiffre – l'argument colonne prend la valeur 1 pour la colonne A, 2 pour la colonne B, etc. Les expressions Cells(2, 5).Select et Range("E2").Select sont donc strictement équivalentes. Lorsque vous enregistrez un déplacement par référence absolue aux cellules, ce déplacement est toujours codé à l'aide de la propriété Range. La propriété Cells offre l'avantage de pouvoir faire référence à des cellules à l'aide de variables numériques – les variables sont traitées au Chapitre 6.

Lorsque vous sélectionnez une plage de cellules, le code Visual Basic généré se présente ainsi :

```
Range("Cell1:Cell2").Select  
Range("Cell_Active").Activate
```

où *Cell1* et *Cell2* représentent respectivement la cellule située à l'angle supérieur gauche de la plage et la cellule située à l'angle inférieur droit de la plage de cellules sélectionnée.

Dans l'expression `Range("Cell_Active").Activate`, l'argument *Cell_Active* indique la cellule active dans la plage sélectionnée. Dans Excel, cette cellule n'est pas noircie et correspond à la cellule à partir de laquelle la sélection a été étendue. Si vous saisissez du texte au clavier, il sera inséré dans cette cellule.

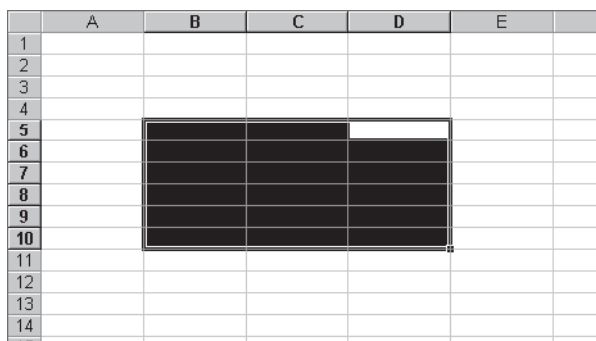
Par exemple, l'expression Visual Basic :

```
Range("B5:D10").Select  
Range("D5").Activate
```

revient à sélectionner une plage dont les cellules situées aux angles supérieur gauche et inférieur droit sont respectivement B5 et D10. Cette sélection a été effectuée en partant de la cellule D5 et en étendant la sélection jusqu'à la cellule B10, si bien que la cellule active de la sélection est la cellule D5 (voir Figure 3.4).

Figure 3.4

La cellule active est la cellule D5.



	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						

Sélection de cellules non contiguës

Vous pouvez sélectionner des cellules non contiguës dans une feuille Excel en maintenant la touche Ctrl enfoncée. La syntaxe de la propriété `Range` lors de la sélection de cellules non contiguës est la suivante :

```
Range("Cell1, Cell2, ..., Celln").Select  
Range("Cell_Active").Activate
```

où les arguments *Cell1*, *Cell2*, ..., *Celln* représentent les cellules successivement sélectionnées. L'expression `Range("Cell_Active").Activate` a le même rôle que lors de la sélection d'une plage de cellules (indique la cellule active de la sélection). L'argument *Cell_Active* représente la dernière cellule sélectionnée.

Par exemple, l'expression Visual Basic :

```
Range("B5, D10, F2, A3").Select  
Range("A3").Activate
```

revient à sélectionner successivement les cellules B5, D10, F2 et A3 en maintenant la touche Ctrl enfoncée.

Sélection de lignes et de colonnes

Lorsque vous sélectionnez une colonne dans une feuille Excel, l'Enregistreur de macro code cette sélection à l'aide de la propriété `Columns`. Dans le cas d'une ligne, c'est la propriété `Rows` qui sera utilisée. Ces propriétés renvoient toutes deux des objets `Range`, de type colonne, pour la propriété `Columns`, et de type ligne, pour la propriété `Rows`.



Pour sélectionner une colonne ou une ligne dans une feuille Excel, cliquez sur l'en-tête de la ligne ou de la colonne que vous souhaitez sélectionner, ou utilisez l'un des raccourcis clavier présentés dans le Tableau 3.1. Pour sélectionner des colonnes ou des lignes contiguës, sélectionnez la première ligne/colonne, puis enfoncez la touche Maj et cliquez sur la dernière ligne/colonne de la plage que vous souhaitez sélectionner.

Lignes contiguës

La syntaxe de la propriété `Rows` est la suivante :

```
Rows("ligne1:ligne2").Select  
Range("cell_active").Activate
```

où les arguments *ligne1* et *ligne2* représentent respectivement l'index de la première et l'index de la dernière ligne de la plage sélectionnée. La méthode `Select` sélectionne l'objet `Range` défini par la propriété `Rows`.

Dans l'expression `Range("cell_active").Activate`, *cell_active* représente l'adresse de la cellule active dans la plage sélectionnée. Cette expression est omise si la cellule active est la cellule située à l'angle supérieur gauche de la plage sélectionnée.



Lorsque vous sélectionnez une plage de lignes dans une feuille Excel, la cellule active est la première cellule de la ligne que vous sélectionnez. Par exemple, si, lors de l'enregistrement d'une macro, vous sélectionnez la ligne 5, puis maintenez la touche Maj enfoncée et sélectionnez la ligne 10, c'est la cellule A5 (située à l'angle supérieur gauche de la plage) qui sera active. En revanche, si vous sélectionnez la ligne 10, puis la ligne 5, en maintenant la touche Maj enfoncée, c'est la cellule A10 qui sera active.

Si la sélection ne porte que sur une ligne, les arguments *ligne1* et *ligne2* ont la même valeur et l'expression `Range("cell_active").Activate` est omise. Par exemple, si vous enregistrez dans une macro la sélection de la ligne 5 de la feuille active, le code Visual Basic correspondant se présentera ainsi :

```
Rows("5:5").Select
```

Figure 3.5

Lorsqu'une seule ligne est sélectionnée, le code ne spécifie pas de cellule active.

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							

Si vous sélectionnez les lignes 5 à 10 (en sélectionnant la ligne 5 en premier, puis la ligne 10, tout en maintenant la touche Maj enfoncée), le code Visual Basic correspondant se présentera ainsi :

```
Rows("5:10").Select
```

Si vous sélectionnez la même plage, mais en sélectionnant d'abord la ligne 10, le code Visual Basic correspondant se présentera ainsi :

```
Rows("5:10").Select
Range("A10").Activate
```


Figure 3.6

*La cellule A10
est la cellule active.*

	A	B	C	D	E	F	G
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

Lorsque vous sélectionnez une ligne, par défaut, la première cellule de la ligne est la cellule active. Si vous modifiez la cellule active – dans ou hors de la plage sélectionnée – en maintenant la touche Ctrl enfoncée et en cliquant sur la cellule que vous souhaitez activer, la propriété Range se substitue à la propriété Rows. Votre code se présente alors ainsi :

```
Range("ligne1:ligne2, cell_active").Select
Range("cell_active").Activate
```

où l'argument *cell_active* représente l'adresse de la cellule active. Par exemple, si vous sélectionnez la ligne 14, puis maintenez la touche enfoncée et cliquez sur la cellule B14 lors de l'enregistrement d'une macro, le code Visual Basic correspondant se présentera ainsi :

```
Range("14:14,B14").Select
Range("B14").Activate
```

Figure 3.7

*La cellule B14
est la cellule active.*

	A	B	C	D	E	F	G
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							

Colonnes contiguës

La syntaxe de la propriété Columns est la même que celle de la propriété Rows :

```
Columns("col1:col2").Select
Range("cell_active").Activate
```

où les arguments *col1* et *col2* représentent respectivement l'index de la première et l'index de la dernière colonne de la plage sélectionnée. La méthode `Select` sélectionne l'objet `Range` défini par la propriété `Columns`.

Dans l'expression `Range("cell_active").Activate`, *cell_active* représente l'adresse de la cellule active dans la plage sélectionnée. Cette expression est omise si la cellule active est la cellule située à l'angle supérieur gauche de la plage sélectionnée.

Si la sélection ne porte que sur une colonne, les arguments *col1* et *col2* ont la même valeur et l'expression `Range("cell_active").Activate` est omise. Par exemple, si vous enregistrez dans une macro la sélection de la colonne B de la feuille active, le code Visual Basic correspondant se présentera ainsi :

```
Columns("B:B").Select
```

Si vous sélectionnez les colonnes B à E (en sélectionnant la colonne B, puis la colonne E, tout en maintenant la touche `Maj` enfoncée), le code Visual Basic correspondant se présentera ainsi :

```
Columns("B:E").Select
```

Si vous sélectionnez la même plage, mais en commençant par sélectionner la colonne E, le code Visual Basic correspondant se présentera ainsi :

```
Columns("B:E").Select  
Range("E1").Activate
```

Lorsque vous sélectionnez une colonne, par défaut, la première cellule de la colonne est la cellule active. Si vous modifiez la cellule active – dans ou hors de la plage sélectionnée – en maintenant la touche `Ctrl` enfoncée et en cliquant sur la cellule que vous souhaitez activer, la propriété `Range` se substitue à la propriété `Columns`. Votre code se présente alors ainsi :

```
Range("col1:col2, cell_active").Select  
Range("cell_active").Activate
```

où l'argument *cell_active* représente l'adresse de la cellule active. Par exemple, si vous sélectionnez la colonne E, puis maintenez la touche `Ctrl` enfoncée et cliquez sur la cellule E5 lors de l'enregistrement d'une macro, le code Visual Basic correspondant se présentera ainsi :

```
Range("E:E,E5").Select  
Range("E5").Activate
```

Figure 3.8

*La cellule E5
est la cellule active.*

	D	E	F	G	H	I	J
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

Info

Vous pouvez substituer la propriété Range aux propriétés Rows et Columns dans le code de votre macro, en conservant les mêmes arguments. Par exemple, les expressions Visual Basic Range("5:10").Select et Rows("5:10").Select correspondent toutes deux à la sélection des lignes 5 à 10 de la feuille active. Lors de l'enregistrement de macros, les sélections sont codées différemment de façon à faciliter la lecture du code.

Lignes et colonnes non contiguës

Lorsque vous sélectionnez des lignes ou des colonnes non contiguës d'une feuille Excel lors de l'enregistrement d'une macro – en maintenant la touche Ctrl enfoncée –, la propriété Range est utilisée. La syntaxe se présente alors ainsi :

```
Range("item1:item2, item3:item4,..., item-n:item-n+1").Select
Range("cell_active").Activate
```

les arguments *item* représentent les index des lignes ou colonnes sélectionnées. Ces arguments vont par paires, chaque paire représentant une plage de lignes ou de colonnes contiguës sélectionnées – le signe : est utilisé comme séparateur. Les arguments item d'une paire peuvent avoir une même valeur.

Si, par exemple, vous sélectionnez les colonnes A, C à E et G et que la cellule active est la cellule G1, la syntaxe Visual Basic représentant cette sélection se présentera ainsi :

```
Range("A:A,C:E,G:G").Select
Range("G1").Activate
```

	A	B	C	D	E	F	G	H	
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									

Figure 3.9

Vous pouvez conjuguer la sélection de colonnes adjacentes et de colonnes non contiguës.

Info

Si, lors de l'enregistrement d'une macro, vous sélectionnez successivement des lignes ou des colonnes contiguës en maintenant la touche Ctrl enfoncée, plutôt que d'utiliser la touche Maj, ces sélections seront considérées comme autonomes et codées comme si les lignes ou les colonnes n'étaient pas contiguës.

Par exemple, si vous cliquez sur l'en-tête de la colonne A, appuyez sur la touche Ctrl et, tout en la maintenant enfoncée cliquez successivement sur les en-têtes des colonnes B, C et E, le code Visual Basic correspondant se présentera ainsi :

```
Range( A:A,B:B,C:C,E:E ).Select
Range("C1").Activate
```

Vous pouvez effectuer la même sélection en utilisant la touche Maj. Cliquez sur l'en-tête de la colonne A, puis appuyez sur la touche Maj et, tout en la maintenant enfoncée, appuyez sur l'en-tête de la colonne C. Relâchez ensuite la touche Maj et appuyez sur la touche Ctrl pour cliquer sur la colonne E. Le code Visual Basic correspondant se présente alors ainsi :

```
Range("A:C,E:E").Select
Range("C1").Activate
```

Vous pouvez, en utilisant les mêmes méthodes de sélection, définir une plage composée de lignes et de colonnes, contiguës ou non. Par exemple, si vous sélectionnez (à l'aide des touches Maj et Ctrl) les colonnes C à E, la colonne G, les lignes 4 à 6 et la ligne 8 comme indiqué à la Figure 3.10, le code de votre macro se présentera ainsi :

```
Range("C:E,G:G,4:6,8:8").Select
Range("A8").Activate
```

Figure 3.10

Vous pouvez conjuguer la sélection de colonnes et de lignes, contiguës ou non.

	A	B	C	D	E	F	G	H	
1									
2									
3									
4									
5									
6									
7									
8									
9									

Enfin, vous pouvez conjuguer la sélection de lignes, colonnes et cellules contiguës ou non. Le code suivant indique la même sélection que dans l'exemple précédent, à laquelle on a ajouté la sélection des cellules A2 à F2 et de la cellule B13.

```
Range("C:E,G:G,4:6,8:8,A2:F2,B13").Select
Range("B13").Activate
```

Référence relative aux cellules

Le codage Visual Basic des déplacements dans Excel par référence relative aux cellules répond aux mêmes principes que le codage du déplacement par référence absolue aux cellules. Cette section présente donc de façon sommaire les spécificités du déplacement par référence relative aux cellules.

Sélection de cellules contiguës

L'expression Visual Basic pour une référence relative à une cellule se présente ainsi :

```
ActiveCell.Offset(RowOffset, ColumnOffset).Range("A1").Select
```

- La propriété `ActiveCell` renvoie un objet `Range` qui représente la cellule active.
- La propriété `Offset` renvoie un objet `Range` (cellule ou plage de cellules), fonction des arguments nommés `RowOffset` et `ColumnOffset`. Ceux-ci indiquent (en nombre de lignes et de colonnes) le décalage à effectuer à partir de la cellule active pour atteindre l'adresse de cet objet.

Ces arguments prennent une valeur numérique – négative si le déplacement s'effectue vers le haut (`RowOffset`) ou vers la gauche (`ColumnOffset`).

- La propriété `Range("A1")` précise qu'il s'agit d'un déplacement de style A1. Autrement dit, la cellule active sert de référence et est virtuellement considérée comme la cellule

A1. Bien que toujours précisée lors de l'enregistrement d'une macro, cette propriété est facultative et vous pouvez la supprimer du code de la macro.

- La méthode `Select` entraîne la sélection de l'objet (ici la cellule) précédemment défini.

Par exemple, l'expression `Visual Basic ActiveCell.Offset(3, 2).Select` entraînera la sélection de la cellule située trois lignes au-dessous et deux colonnes à droite de la cellule active.

L'expression `ActiveCell.Offset(-3, -2).Select` entraînera la sélection de la cellule située trois lignes au-dessus et deux colonnes à gauche de la cellule active.

Lorsque vous sélectionnez une plage de cellules, le code Visual Basic généré se présente ainsi :

```
ActiveCell.Offset(RowOffset, ColumnOffset).Range("A1:Cell12").Select
```

où `Offset(RowOffset, ColumnOffset)` indique le déplacement à effectuer relativement à la cellule active et `A1:Cell12` représente la plage de cellules sélectionnées en estimant que la cellule active est A1.

Par exemple, l'expression Visual Basic

```
ActiveCell.Offset(-3, 0).Range("A1:A4").Select
```

revient à étendre la sélection de la cellule active jusqu'à la cellule située trois lignes au-dessus. On obtient alors une sélection équivalente à la sélection A1:A4 (il peut s'agir de B1:B4, C5:C8, etc.)

Sélection de cellules non contiguës

Vous pouvez sélectionner des cellules non contiguës dans une feuille Excel en maintenant la touche `Ctrl` enfoncée. La syntaxe de la propriété `Range` lors de la sélection de cellules non contiguës est la suivante :

```
ActiveCell.Range("A1, Cell12,..., Celln").Select
```

où A1 représente la cellule active et les arguments `Cell12,..., Celln` représentent les adresses des cellules successivement sélectionnées, par position relative à la cellule initiale. L'argument `Cell_Active` représente la dernière cellule sélectionnée.

Info

Lors de déplacements par référence relative aux cellules, le codage Visual Basic se fait par style de référence A1 ; cela signifie que l'adresse virtuelle A1 est toujours attribuée à la cellule active, ce qui permet de représenter les déplacements et sélections d'autres cellules par rapport à cette adresse virtuelle.

Par exemple, l'expression Visual Basic

```
ActiveCell.Range("A1,A7,C7,C1").Select  
ActiveCell.Offset(0, 2).Range("A1").Activate
```

indique que la macro sélectionnera simultanément la cellule active (virtuellement A1) et les cellules dont les adresses virtuelles (en style de référence A1) sont A7, C7, C1. La cellule active sera la cellule située deux colonnes à droite – `Offset(0, 2)` – de la cellule initialement active (virtuellement C1).

Si la cellule active au moment de l'exécution de la macro est la cellule B5, les cellules B5, B11, D11 et D5 seront sélectionnées. La cellule D5 sera la cellule active.

Pour bien comprendre ce principe, gardez à l'esprit que ce qui fait la relativité du déplacement est le repère d'origine. Les adresses des cellules sont déterminées par l'adresse de la cellule servant de repère.

Redimensionner une plage

Pour modifier l'ampleur d'une plage de cellules, utilisez la méthode `Resize` selon la syntaxe suivante :

```
expression.Resize(RowSize, ColumnSize)
```

où *expression* renvoie l'objet `Range` à redimensionner. *RowSize* et *ColumnSize* sont facultatifs et renvoient respectivement le nombre de lignes et le nombre de colonnes qui doivent être ajoutées (valeurs positives) ou retirées (valeurs négatives) à la sélection. Si l'un ou l'autre de ces arguments est omis, il prend la valeur par défaut de 0 et la dimension correspondante n'est pas modifiée.

L'instruction suivante étend la sélection en cours d'une ligne et d'une colonne supplémentaires. Elle utilise pour cela la propriété `Count` qui, appliquée à la collection `Rows`, renvoie le nombre de lignes et, appliquée à la collection `Columns`, renvoie le nombre de colonnes.

```
Selection.Resize(Selection.Rows.Count + 1, Selection.Columns.Count + 1).Select
```

Référence aux cellules fonction de leur contenu

Vous pouvez effectuer ou modifier une sélection en fonction du contenu des cellules. Il sera ainsi utile d'étendre la sélection à l'ensemble des cellules non vides d'un tableau avant d'effectuer un tri ou d'identifier la première cellule vide dans une colonne afin d'y insérer du contenu. VBA propose pour cela les propriétés suivantes :

- **CurrentRegion.** Retourne la plage de cellules courante (les cellules contiguës qui contiennent des données).

- **End.** Permet d'identifier les cellules vides dans une plage de cellules.
- **UsedRange.** Retourne une plage composée des cellules d'une feuille contenant des données.

La propriété *CurrentRegion*

Pour étendre la sélection à la zone courante, c'est-à-dire à la zone entourée par une combinaison de lignes et de colonnes vides, vous utiliserez la propriété *CurrentRegion* selon la syntaxe suivante :

```
Range.CurrentRegion.Select
```

où *Range* est une expression qui renvoie un objet *Range*. Considérez les deux instructions suivantes :

```
ActiveCell.CurrentRegion.Select  
Range("C8").CurrentRegion.Select
```

La première instruction étend la sélection à la zone courante à partir de la cellule active, tandis que la seconde procède de même, mais à partir de la cellule C8.

À la Figure 3.11, la sélection a été étendue à la zone courante à partir de la cellule C8.

	A	B	C	D	E	F	G	H	I	J	K
1			traffic figures from 26th february to 5th march 2002								
2											
3											
4											
5			26-févr	DE	FR	UK	ES	IT			
6			27-févr	22 480	52 634	5 922	98 498	52 634			
7			28-févr	34 423	46 889	4 533	45 867	46 889			
8			01-mars	32 279	40 978	4 188	27 678	40 978			
9			02-mars	29 628	51 158	3 267	87 649	51 158			
10			03-mars	30 877	33 545	5 168	68 795	33 545			
11			04-mars	26 647	31 143	5 056	57 869	31 143			
12			05-mars	22 500	34 837	5 777	40 985	34 837			
13				25 800	36 885		36 885	36 885			
14			Total	224 634	328 069	33 911					
15			average per day	28 079	41 009	4 844					
16			average per month	842 378	1 230 259	145 333					
17											
18											
19											
20			Traffic increase evaluation								
21											
22			30% more with portal integration	1 120 362	1 636 244	193 293					
23											
24											
25			20% more for new countries and new layout	1 344 434	1 963 493	231 951					
26											
27											
28			10% more directories and search engines promotion	1 478 878	2 159 842	255 146					

Figure 3.11
La propriété CurrentRegion permet de "capturer" une zone contenant des données à partir d'une cellule.

La propriété *End*

La propriété *End* renvoie un objet *Range* qui représente la dernière cellule d'une zone. Cela revient à employer dans un tableau Excel la combinaison clavier Fin+flèche de direction. Utilisez la propriété *End* selon la syntaxe suivante :

```
Range.End(Direction)
```

où *Range* renvoie l'objet *Range* à partir duquel on recherche la dernière cellule, et où *Direction* représente le sens dans lequel on se déplace. Il peut s'agir de l'une des constantes *xlDirection* suivantes :

- **xlDown**. Déplacement vers le bas.
- **xlToRight**. Déplacement vers la droite.
- **xlToLeft**. Déplacement vers la gauche.
- **xlUp**. Déplacement vers le haut.

Appliquées au tableau de la Figure 3.11, les quatre instructions suivantes renvoient respectivement l'objet *Range* représentant les cellules E4, E12, C8 et H8.

```
Range("E8").End(xlUp).select      'renvoie la cellule E4
Range("E8").End(xlDown).select    'renvoie la cellule E12
Range("E8").End(xlToLeft).select  'renvoie la cellule C8
Range("E8").End(xlToRight).select 'renvoie la cellule H8
```

Vous pouvez évidemment utiliser la propriété *End* pour sélectionner des plages de cellules, comme nous l'avons vu précédemment dans ce chapitre. Considérez les exemples suivants.

- Sélection d'une plage de la première à la dernière cellule non vide d'une colonne :

```
Range("A1", Range("A1").End(xlDown)).Select
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

- Sélection d'une plage de la dernière cellule non vide jusqu'à la première :

```
Range("A32", Range("A32").End(xlUp)).Select
Range(ActiveCell, ActiveCell.End(xlUp)).Select
```

Si vous souhaitez sélectionner la première cellule vide d'une zone plutôt que la dernière cellule non vide, utilisez la propriété *Offset* pour décaler la sélection. La première instruction, ci-après, sélectionne la première cellule vide au bas de la colonne, tandis que la seconde sélectionne la première cellule vide à droite :

```
Range("A1").End(xlDown).Offset(1, 0).Select
Range("A1").End(xlToRight).Offset(1, 0).Select
```

La propriété *UsedRange*

La propriété *UsedRange* retourne la plage de cellules contenant les données d'une feuille. Cette propriété est donc particulièrement pratique pour identifier les cellules sur lesquelles doivent s'appliquer des traitements lors de l'exécution d'un programme. La plage retournée par *UsedRange* est une plage de cellules contiguës dont les limites sont définies par :

- la cellule dont l'index de colonne est le plus élevé ;
- la cellule dont l'index de colonne est le plus faible ;
- la cellule dont l'index de ligne est le plus élevé ;
- la cellule dont l'index de ligne est le plus faible.

Figure 3.12
*La propriété *UsedRange* retourne une plage qui englobe toutes les cellules contenant des données.*

A	B	C	D	E	F	G	H
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							
31							
32							
33							
34							
35							
36							
37							
38							
39							
40							

L'instruction suivante sélectionne la plage de cellules utilisée sur la feuille active du classeur actif :

```
ActiveWorkbook.ActiveSheet.UsedRange.Select
```

Référence aux plages de cellules nommées

Pour faire référence à une plage de cellules nommée dans Excel, utilisez la syntaxe suivante :

```
Range(" [NomClasseur]NomFeuille!NomPlage")
```

où *NomClasseur* est le nom du classeur, et *NomFeuille* celui de la feuille qui contient la plage nommée *NomPlage*.

L'exemple suivant passe en gras les cellules de la plage nommée MaPlage, située sur la feuille Feuil1 du classeur Test.xlsx :

```
Range("[Test.xlsx]Feuil1!MaPlage").Font.Bold = True
```

Pour sélectionner une plage nommée, utilisez la méthode GoTo qui active successivement le classeur et la feuille si nécessaire, puis sélectionne la plage voulue. Les deux instructions suivantes sélectionnent la plage nommée MaPlage, puis en effacent le contenu :

```
Application.Goto Reference:="[Test.xlsx]Feuil1!MaPlage"  
Selection.ClearContents
```



4

Découvrir Visual Basic Editor

Au sommaire de ce chapitre

- Accéder à Visual Basic Editor
- Les outils et les fenêtres de Visual Basic Editor
- Paramétrer Visual Basic Editor

Visual Basic Editor est l'environnement de développement intégré de VBA. C'est dans cet environnement que vous passerez l'essentiel de votre temps lors du développement de projets VBA. Les chapitres précédents vous ont fait découvrir la fenêtre Code de Visual Basic Editor à travers la modification et la création de macros. Mais Visual Basic Editor ne se résume pas à un simple éditeur de code. Il s'agit d'un logiciel complet, proposant des outils d'aide au développement que ce chapitre vous propose de découvrir.

Accéder à Visual Basic Editor

Lorsque vous choisissez de modifier une macro existante ou de créer une nouvelle macro, selon les procédures étudiées au Chapitre 2, vous accédez à la fenêtre Code de Visual Basic Editor. Vous pouvez aussi développer un projet VBA en accédant directement à Visual Basic Editor, sans passer par la boîte de dialogue Macro.

On accède toujours à Visual Basic Editor à partir d'une application hôte. Autrement dit, une session Visual Basic Editor peut être liée à Word, PowerPoint ou encore Excel, mais ne permet d'accéder qu'aux projets de l'application à partir de laquelle il a été exécuté.



Lorsque vous êtes dans Visual Basic Editor, vous pouvez accéder à l'ensemble des éléments constitutifs des projets accessibles, y compris aux macros disponibles dans la boîte de dialogue Macro.



Pour qu'un projet soit accessible dans Visual Basic Editor, il faut que le document dans lequel il est stocké soit ouvert dans l'application hôte.

Pour accéder à Visual Basic Editor à partir d'Excel 2007, activez l'onglet Développeur du ruban, puis sélectionnez Visual Basic Editor, ou tapez le raccourci clavier Alt+F11.



Pour accéder à Visual Basic Editor à partir d'une version d'Excel antérieure à 2007, sélectionnez Outils > Macro > Visual Basic Editor, ou tapez le raccourci clavier Alt+F11. Vous pouvez également accéder à Visual Basic Editor à partir du bouton Visual Basic Editor de la barre d'outils Visual Basic, si celle-ci s'affiche.

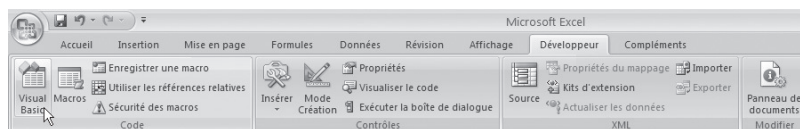
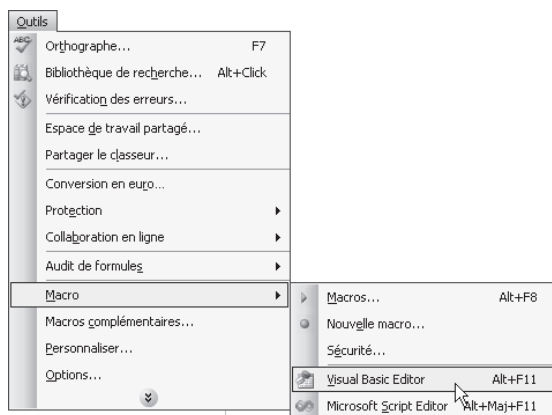


Figure 4.1

Nouvelle interface d'Office 2007 oblige, l'accès à Visual Basic Editor se fait via le ruban.

Figure 4.2

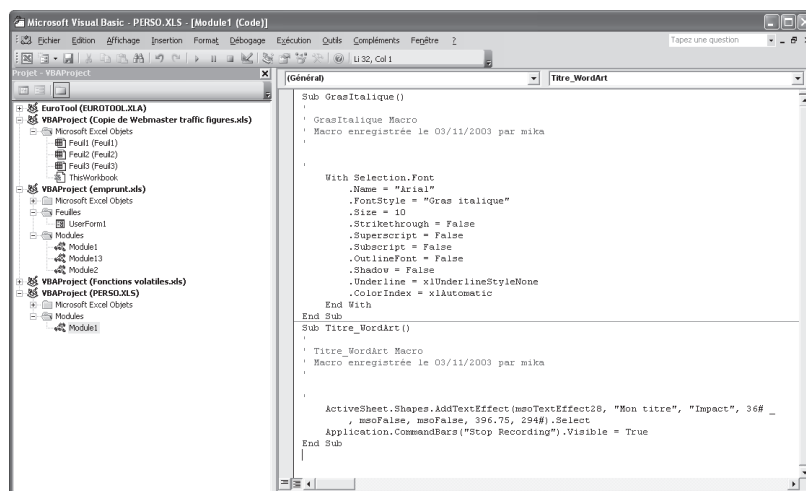
Pour accéder à Visual Basic Editor à partir d'une version d'Excel antérieure à 2007, sélectionnez la commande... Visual Basic Editor !



La Figure 4.3 présente la fenêtre de Visual Basic Editor. Il se peut que, sur votre ordinateur, la fenêtre ne présente pas les mêmes éléments. Vous verrez par la suite comment afficher les différents composants de Visual Basic Editor.

Figure 4.3

La fenêtre de Visual Basic Editor.



Conseil

Dans bien des cas, l'enregistrement de macros reste la méthode la plus rapide et la plus sûre pour démarrer vos projets VBA. En laissant à l'Enregistreur de macro le soin de convertir les actions exécutées en code Visual Basic, vous êtes assuré de ne pas commettre d'erreur de saisie.

Il se peut cependant que la première étape de développement de votre projet consiste à créer une feuille permettant une interaction avec l'utilisateur. Vous accédez alors directement à Visual Basic Editor, sans passer par la boîte de dialogue Macro.

Pour quitter Visual Basic Editor et retourner à l'application hôte, vous pouvez :

- ouvrir le menu Fichier, sélectionner la commande Fermer et retourner à Microsoft Excel ;
- taper le raccourci clavier Alt+Q ;
- cliquer sur la case de fermeture de Visual Basic Editor (située à l'extrémité supérieure droite de la fenêtre).

Pour retourner à l'application hôte sans quitter Visual Basic Editor, vous pouvez :



- cliquer sur l'icône Affichage Microsoft Excel, située à l'extrême gauche de la barre d'outils Standard ;
- taper le raccourci clavier Alt+F11.

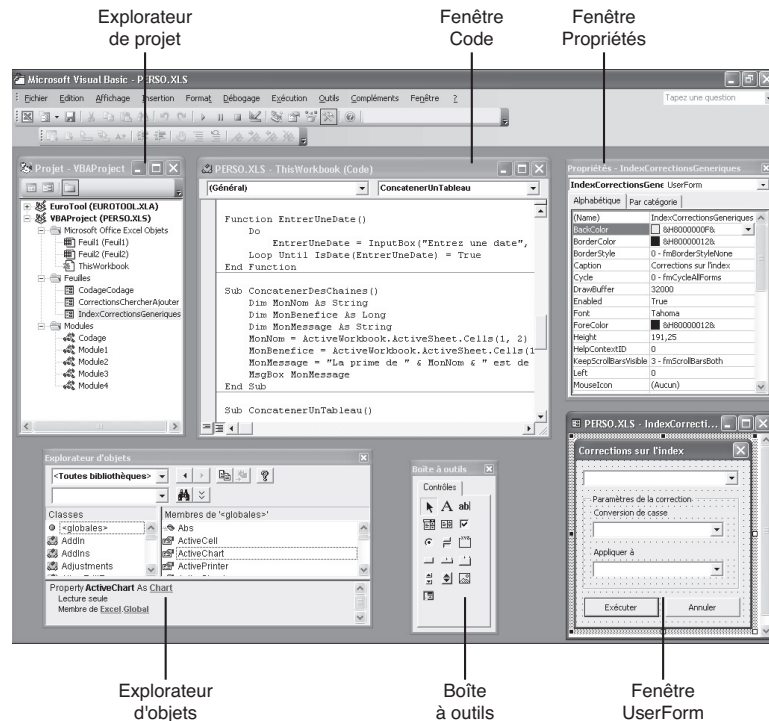
Les outils et les fenêtres de Visual Basic Editor

Cette section présente sommairement les éléments essentiels de l'interface de Visual Basic Editor. Vous en découvrirez plus précisément les fonctionnalités au fur et à mesure que vous avancerez dans la lecture de l'ouvrage.

Les éléments essentiels de l'interface de Visual Basic Editor sont :

- **L'Explorateur de projet.** Il permet de visualiser les différents projets et éléments constitutifs qui les composent – objets, modules, modules de classe, feuilles (ou formulaires) et Référence –, et d'accéder à ces éléments ou au code qui leur est attaché. Pour qu'un projet apparaisse dans l'Explorateur de projet, il faut que le document auquel il est attaché soit ouvert dans l'application hôte.
- **La fenêtre Propriétés.** Elle permet de visualiser et de modifier l'ensemble des propriétés associées aux objets constitutifs d'un projet.
- **La fenêtre Code.** Vous pouvez y éditer le code de vos projets. Visual Basic Editor propose des aides à l'écriture de code et des outils de débogage.
- **La fenêtre UserForm et la boîte à outils.** La fenêtre UserForm est l'espace dans lequel vous concevez les feuilles VBA. La boîte à outils propose des contrôles communs tels que des cases à cocher ou des zones de listes déroulantes que vous pouvez placer sur une feuille qui constituera une interface pour votre application.
- **L'Explorateur d'objets.** Il référence les classes, propriétés, méthodes, événements et constantes disponibles dans les bibliothèques d'objets et les procédures de votre projet. Il permet de rechercher et d'utiliser des objets que vous créez, ainsi que des objets provenant d'autres applications.

Figure 4.4
Visual Basic Editor.



L'Explorateur de projet

L'Explorateur de projet permet d'explorer les différents projets chargés dans l'application hôte et les éléments qui les composent. À partir de l'Explorateur de projet, vous pouvez accéder à n'importe quel élément constitutif d'un projet, y ajouter de nouveaux éléments ou, au contraire, en supprimer.

Afficher et masquer l'Explorateur de projet

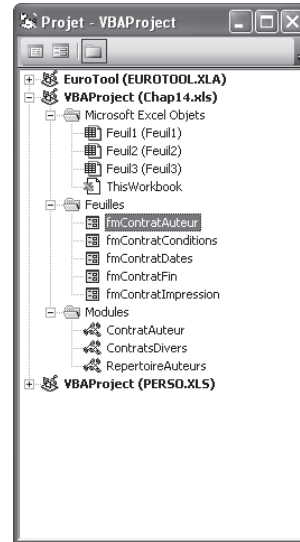
Pour afficher l'Explorateur de projet, vous pouvez :

- choisir la commande Explorateur de projet du menu Affichage ;
- taper le raccourci clavier Ctrl+R ;
- cliquer sur le bouton Explorateur de projet de la barre d'outils Standard de Visual Basic Editor.



Figure 4.5

L'Explorateur de projet permet d'accéder aisément aux différents éléments constitutifs d'un projet.

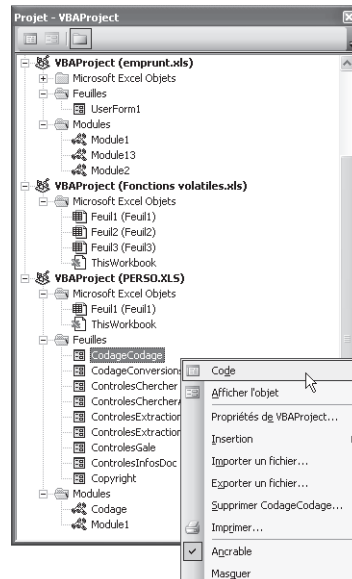


Pour masquer l'Explorateur de projet, vous pouvez :

- cliquer du bouton droit dans sa fenêtre et sélectionner la commande Masquer du menu contextuel qui s'affiche ;

Figure 4.6

Le menu contextuel de l'Explorateur de projet.



- cliquer du bouton droit sur sa barre de titre et sélectionner la commande Fermeture ;
- taper le raccourci clavier Alt+F4 ;
- cliquer sur la case de fermeture de la fenêtre.

Naviguer dans l'Explorateur de projet

À l'instar de l'Explorateur de Windows 95, l'Explorateur de projet présente les différents projets et leurs éléments constitutifs de façon hiérarchique. Au premier niveau de l'Explorateur, apparaissent les différents projets. Sous chaque projet se trouvent des dossiers contenant chacun des éléments spécifiques :

- **Microsoft Excel Objets.** Contient les documents associés au projet. Il s'agit du classeur dans lequel est stocké le projet, et de ses feuilles de calcul. Vous verrez au Chapitre 15 que vous pouvez associer des procédures spécifiques à ces objets de façon à contrôler les interventions d'un utilisateur sur un classeur.
- **Feuilles.** Contient les feuilles (ou formulaires) du projet. Vous apprendrez à créer des feuilles aux Chapitres 12 à 14.
- **Modules.** Contient les modules standard (ou modules de code) – tels que les macros – constitutifs d'un projet.
- **Modules de classe.** Contient les éventuels modules de classe d'un projet.
- **Références.** Contient les références à d'autres projets.



Si un projet ne contient aucun module ou module de classe, les dossiers correspondants n'apparaîtront pas sous le projet dans l'Explorateur de projet.



Lorsque vous enregistrez ou créez une macro, elle est enregistrée dans un nouveau module accessible via le dossier Modules du projet correspondant. Le nom de ce module est Module1, Module2 si Module1 existe déjà, etc.

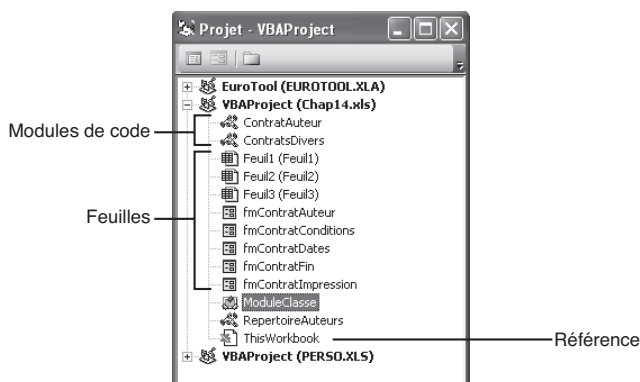
Le signe plus (+) permet de développer l'arborescence d'un dossier ou d'un projet, et le signe moins (–) de la réduire. À la Figure 4.6, l'arborescence du dossier Feuilles du projet PERSONAL.XLSB est développée, tandis que les autres dossiers ainsi que le projet ChiffresRepresentants.xlsx sont réduits.



Plutôt qu'un affichage par dossier, les éléments constitutifs d'un projet peuvent être présentés par ordre alphabétique dans l'Explorateur de projet. Cliquez simplement sur le bouton Basculer dossiers de l'Explorateur de projet. La Figure 4.7 présente l'Explorateur de projet après masquage des dossiers. Pour revenir à un affichage par dossier, cliquez de nouveau sur ce bouton.

Figure 4.7

Lorsque l'affichage des dossiers est désactivé, les icônes permettent de distinguer les éléments du projet.



Accéder aux objets et au code des projets

Outre le bouton Basculer dossiers, l'Explorateur de projet présente deux boutons permettant un accès simple et rapide au code et aux objets constitutifs d'un projet :



Afficher le code

Affiche le code de l'élément sélectionné dans l'Explorateur de projet afin d'en permettre l'écriture ou la modification.



Afficher l'objet

Affiche l'objet sélectionné dans l'Explorateur de projet. Il peut s'agir d'une feuille (dossier UserForm) ou d'un document. Ce bouton est désactivé si l'objet sélectionné est un module de code.

L'Explorateur d'objets

Lorsqu'on commence à développer en VBA, la difficulté essentielle consiste à manipuler les objets de l'application hôte (dans notre cas les objets Excel). Comment, par exemple, accéder à une plage de cellules d'une feuille spécifique d'un classeur Excel et y insérer une formule ? L'Enregistreur de macros est dans de nombreux cas la solution à ce problème. Vous manipulez les objets Excel après avoir activé l'Enregistreur de macro, puis vous visualisez dans Visual Basic Editor les mots clés Visual Basic utilisés pour accéder aux objets, à leurs propriétés et à leurs méthodes.

Cependant, certains éléments de code d'un programme VBA ne peuvent être générés à l'aide de l'Enregistreur de macro et ne peuvent qu'être saisis dans la fenêtre Code du programme. Vous devez alors connaître la position de l'objet auquel vous souhaitez accéder dans la hiérarchie de classes de l'application. Vous devez aussi connaître les méthodes et propriétés associées à cet objet pour pouvoir le manipuler ou en extraire des informations.

Les chapitres précédents vous ont initié à la syntaxe VBA permettant d'accéder à un objet. Pour autant, lorsque vous commencerez à développer dans Visual Basic Editor, vous ne connaîtrez pas toujours le chemin à emprunter pour accéder à tel ou tel objet, ni la méthode à lui appliquer pour effectuer telle ou telle opération. L'Explorateur d'objets constitue pour cela une aide très appréciable pour le développeur. L'Explorateur d'objets recense en effet l'ensemble des objets disponibles dans les *bibliothèques d'objets* accessibles pour un projet, ainsi que les propriétés, constantes, méthodes et événements associés à ces objets. Il constitue en cela un très supplément à l'aide de Visual Basic pour Applications.



*Une bibliothèque d'objets est un fichier contenant toutes les données d'objets (leurs propriétés, méthodes, événements, constantes, etc.). Ce fichier porte l'extension .OLB et c'est à lui que se réfère Visual Basic lorsque vous manipulez des objets Excel. Le nom de fichier de la bibliothèque d'objets d'Excel ainsi que son emplacement varient d'une version à l'autre. Pour le localiser, effectuez une recherche sur *.olb.*

Lorsque vous recherchez un objet ou souhaitez en connaître les membres – c'est ainsi que l'on nomme les éléments Visual Basic (méthodes, propriétés, événements, constantes) associés à un objet –, l'Explorateur d'objets vous fournit une documentation complète. Il permet d'accéder au modèle d'objets de l'application hôte, mais aussi aux objets d'autres applications et aux objets, procédures et constantes que vous avez créés dans le cadre de votre projet, ainsi qu'aux rubriques d'aide associées à chacun de ces éléments.

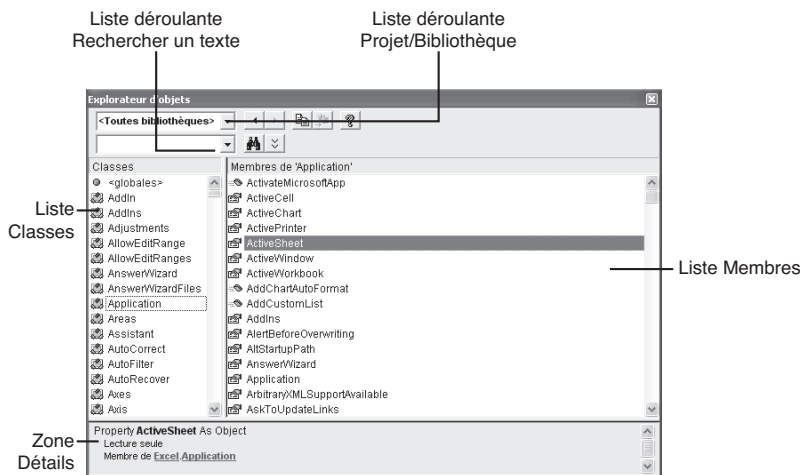
Afficher et masquer l'Explorateur d'objets

Pour afficher l'Explorateur d'objets, vous pouvez :

- sélectionner la commande Explorateur d'objets du menu Affichage ;
- taper le raccourci clavier F2 ;
- cliquer sur le bouton Explorateur d'objets de la barre d'outils Standard de Visual Basic Editor.



Figure 4.8
L'Explorateur d'objets permet d'explorer l'ensemble des objets disponibles pour un projet.



Pour masquer l'Explorateur d'objets, vous pouvez :

- cliquer du bouton droit dans sa fenêtre et sélectionner la commande Masquer du menu contextuel qui s'affiche ;
- cliquer sur l'icône située à gauche de la barre de titre et sélectionner la commande Fermeture ;
- cliquer sur la case de fermeture de la fenêtre.

Naviguer dans l'Explorateur d'objets

S'il peut effrayer le programmeur novice, l'Explorateur d'objets est en réalité d'une utilisation simple et intuitive. Cette section en présente l'utilisation.

La zone de liste Projet/Bibliothèque

La zone de liste déroulante Projet/Bibliothèque permet de sélectionner le projet ou la bibliothèque d'objets de votre choix. Le Tableau 4.1 présente les bibliothèques les plus courantes accessibles dans l'Explorateur d'objets.

Tableau 4.1 : Les bibliothèques les plus courantes de l'Explorateur d'objets

Bibliothèque	Description
<Toutes bibliothèques>	Lorsque cette option est sélectionnée, les objets sont affichés, toutes bibliothèques confondues.
MSForms	Contient les objets accessibles dans la fenêtre UserForm, tels que les boutons d'options, cases à cocher, zones de liste, etc.

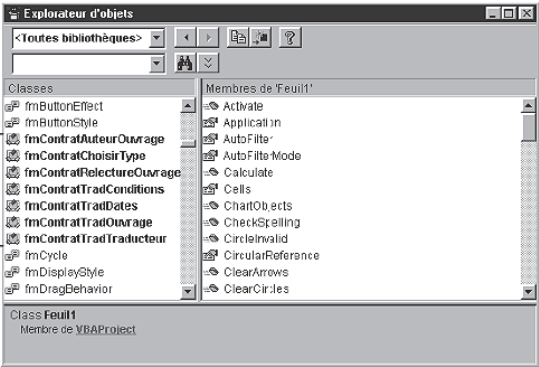
<i>Bibliothèque</i>	<i>Description</i>
Office	Contient les objets Microsoft Office. Il s'agit des objets communs aux applications Office, tels que l'objet Balloon représentant l'info-bulle dans laquelle le Compagnon Office affiche les informations.
VBA	Il s'agit de la bibliothèque Visual Basic pour Applications. Les objets y sont classés par thème. Par exemple, le module Information contient les procédures permettant de renvoyer et de vérifier des informations, le module String contient les procédures permettant d'effectuer des opérations sur des chaînes.
Excel	Contient les objets d'Excel.
Autres applications	Contient les objets des autres applications référencées dans votre projet. Référencer une autre application permet d'accéder aux objets de cette application à partir d'Excel. Vous verrez au Chapitre 6 comment créer une référence à la bibliothèque d'objets d'une autre application, et comment manipuler des objets de cette dernière.
Projets	Affiche les objets propres au projet, tels que les feuilles, les modules de classe et les modules de code que vous avez créés.

Les zones Classes et Membres de

Lorsque vous sélectionnez une bibliothèque dans la zone de liste Bibliothèque/Projet, la zone Classes affiche l'ensemble des classes disponibles dans cette bibliothèque. Les classes sont affichées par type et par ordre alphabétique au sein de chaque type. Chaque type de classe (feuille, module de code, module de classe) est symbolisé par une icône. Lorsqu'une classe contient du code rédigé par l'utilisateur, son nom apparaît en gras (voir Figure 4.9).

Figure 4.9
Les classes contenant du code apparaissent en gras.

Ces classes comportent du code rédigé par l'utilisateur



Pour explorer les membres d'une classe, sélectionnez cette classe dans la zone Classes. Les membres disponibles pour cette classe apparaissent dans la zone Membres de "Classe" – où Classe est le nom de la classe sélectionnée.



On appelle membres d'une classe l'ensemble des éléments référencés pour cette classe, c'est-à-dire ses propriétés, constantes, méthodes et événements.

Les membres affichés dans la zone Membres de "Classe" sont affichés par type (propriétés, constantes, méthodes et événements) et par ordre alphabétique au sein de chaque type. Chaque type de membre est symbolisé par une icône. Lorsqu'un membre contient du code rédigé par l'utilisateur, son nom apparaît en gras.



Pour afficher les membres d'une classe par ordre alphabétique, indépendamment de leur type, cliquez du bouton droit dans l'Explorateur d'objets et, dans le menu contextuel qui s'affiche, sélectionnez la commande Membres du groupe. Pour revenir à un affichage par groupe, répétez cette opération.

Accéder à la rubrique d'aide de l'objet sélectionné

Lorsqu'un élément est sélectionné dans l'Explorateur d'objets, il est très simple d'accéder à la rubrique d'aide qui lui est affectée. Vous pouvez :

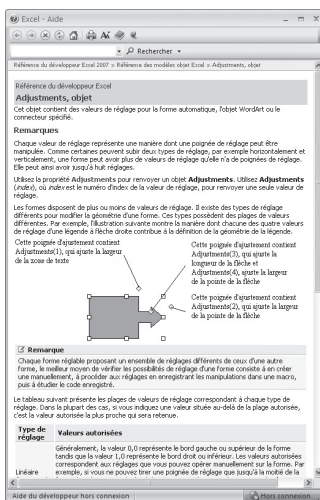


- cliquer sur le bouton Aide de l'Explorateur d'objets ;
- taper le raccourci clavier F1 ;
- cliquer du bouton droit sur l'élément voulu et sélectionner la commande Aide du menu contextuel.

Vous obtenez ainsi une aide précieuse sur l'utilisation de l'élément voulu (voir Figure 4.10). Vous pouvez éventuellement consulter l'exemple de code fourni dans l'aide de VBA et en copier la syntaxe afin de la coller dans votre propre code.

Figure 4.10

L'Explorateur d'objets permet d'accéder rapidement aux rubriques d'aide des objets affichés.



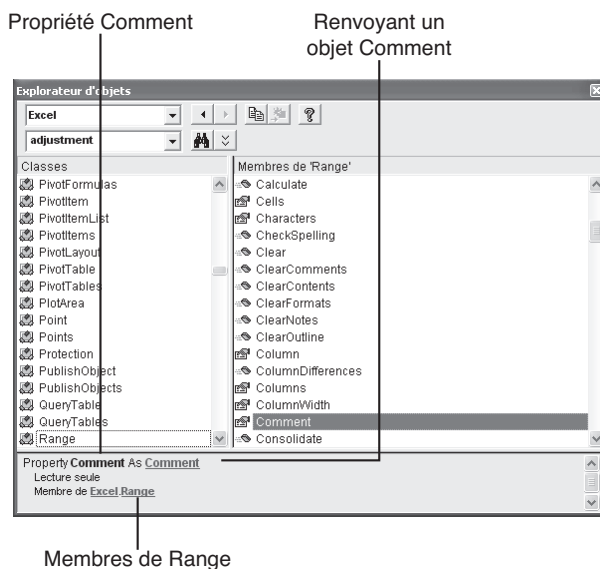
Les autres contrôles de l'Explorateur d'objets

La zone Détails affiche un bref descriptif du membre sélectionné dans la zone Membres de (voir Figure 4.11). Si aucun membre n'est sélectionné, la description concerne la classe sélectionnée dans la zone Classes. Enfin, si aucune classe n'est sélectionnée, cette zone indique le chemin d'accès à la bibliothèque ou au projet affiché dans l'Explorateur d'objets.

Cette zone présente aussi, sous la forme d'un hyperlien de couleur verte, la position de l'objet sélectionné dans le modèle d'objets, et les éléments du langage éventuellement liés à la syntaxe de cet objet. Pour afficher les membres d'un objet apparaissant en hyperlien dans la zone Membres de, cliquez sur l'hyperlien le représentant.

Figure 4.11

La zone Détails récapitule les données propres à l'élément sélectionné.



Le bouton Retourner permet de revenir aux sélections précédentes dans les listes Classes et Membres de. Chaque clic vous fait remonter d'une sélection. Le bouton Avancer permet de revenir aux dernières sélections effectuées après avoir utilisé le bouton Retourner.

Le bouton Copier place dans le Presse-papiers le texte sélectionné. Il peut s'agir du texte de la zone Détails, de la zone Membres de ou de la zone Classes. Le texte ainsi copié peut être collé dans une fenêtre Code.

Le bouton Afficher la définition est accessible lorsque la classe ou le membre de classe sélectionné contient du code rédigé. Il entraîne l'affichage de la fenêtre Code correspondant à la sélection en cours dans l'Explorateur d'objets.

Rechercher du texte

L'Explorateur d'objets offre un outil permettant de rechercher des chaînes de caractères dans les bibliothèques de votre choix. Pour rechercher du texte dans les bibliothèques/projets, procédez comme suit :

1. Dans la zone Bibliothèque/Projet, sélectionnez la bibliothèque ou le projet dans lesquels vous souhaitez que la recherche s'effectue.

Si vous ne savez dans quelle bibliothèque effectuer la recherche, sélectionnez <Toutes bibliothèques>.

2. Dans la zone Rechercher texte, saisissez le texte à rechercher.



Si vous n'êtes pas certain de l'orthographe du texte à rechercher, utilisez les caractères génériques suivants :

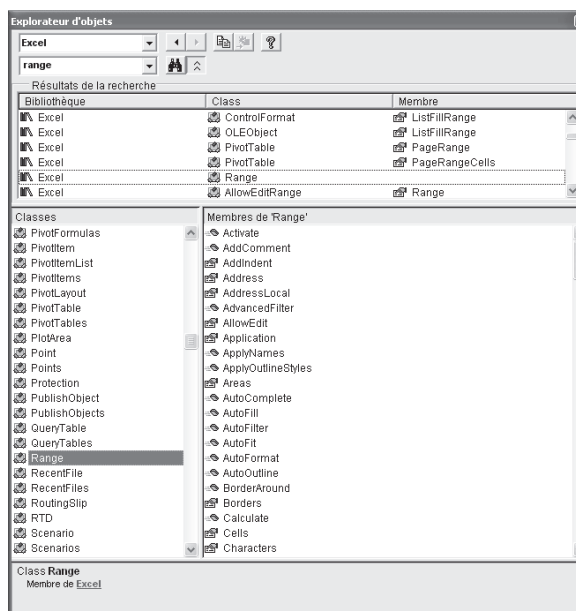
- * toute chaîne ;
- ? tout caractère.



3. Cliquez sur le bouton Rechercher. La zone Résultats de la recherche s'affiche. Pour chacun des éléments trouvés, la bibliothèque, la classe et le membre sont indiqués (voir Figure 4.12).

Figure 4.12

La zone Résultat de la recherche liste l'ensemble des éléments trouvés contenant la chaîne spécifiée.

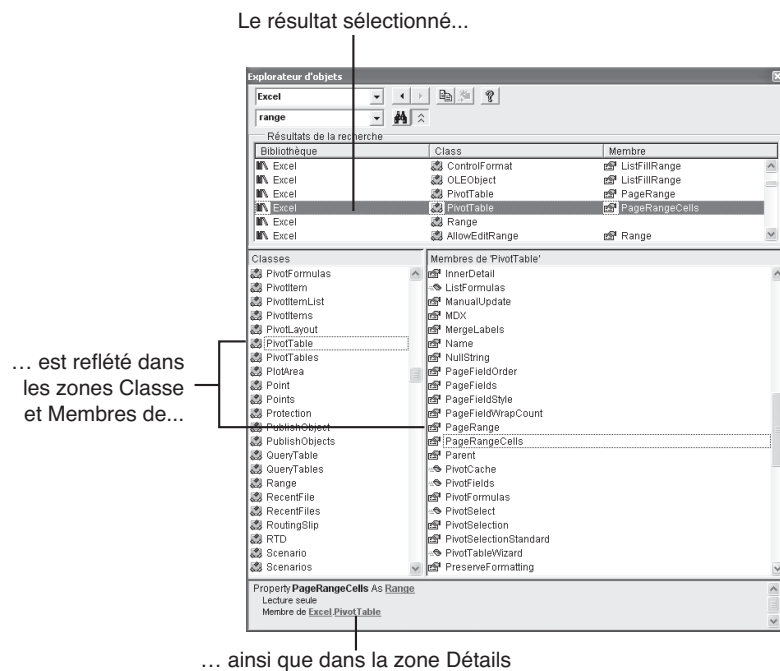


4. Si vous souhaitez afficher les résultats pour d'autres bibliothèques, modifiez simplement la sélection dans la zone Bibliothèque/Projet. Les résultats de la recherche sont automatiquement mis à jour.
5. Pour afficher les informations concernant l'un des résultats de la recherche dans les zones Classe et Membres de, sélectionnez l'élément voulu dans la zone Résultat de la recherche.

Les zones Classe et Membres de sont automatiquement mises à jour, les objets concernés par la sélection étant entourés de pointillés (voir Figure 4.13).

Figure 4.13

Sélectionnez l'élément qui vous intéresse dans la zone Résultat de la recherche.



6. Pour ouvrir la rubrique d'aide associée à l'un des éléments trouvés, sélectionnez-le et cliquez sur le bouton Aide, ou tapez le raccourci clavier F1.



7. Votre recherche terminée, vous pouvez choisir de masquer la zone Résultats de la recherche. Pour cela, cliquez simplement sur le bouton Afficher/Masquer les résultats de la recherche.

Pour afficher de nouveau cette zone sans lancer une nouvelle recherche, cliquez une nouvelle fois sur le bouton.

La fenêtre UserForm

La fenêtre UserForm permet de dessiner des boîtes de dialogue pour vos projets. Dans Visual Basic pour Applications, ces boîtes de dialogue sont appelées des *feuilles* – on parle aussi de *formulaire*s. Une feuille peut être très simple, ou présenter un grand nombre de fonctionnalités. La Figure 4.14 présente une feuille créée dans Visual Basic Editor.

Figure 4.14

Une feuille développée dans Visual Basic Editor.

Dates de remise et de parution

Indiquez la date à valider et sélectionnez-la sur le calendrier.

Date de remise :

Date de parution :

Date à valider :

December 2003

lun.	mar.	mer.	jeu.	ven.	sam.	dim.
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Le développement de feuilles est un aspect essentiel de la programmation VBA. Les Chapitres 12 à 14 sont entièrement consacrés à ce sujet. Cette section présente sommairement les possibilités de développement dans la fenêtre UserForm.

On peut distinguer deux phases essentielles dans la création d'une interface utilisateur :

- le développement visuel de la feuille – traité au Chapitre 12 ;
- l'association de code aux différents éléments de la feuille – traitée aux Chapitres 13 et 14.

La première phase consiste à *dessiner* la feuille, c'est-à-dire à créer l'interface dont vous avez besoin pour votre application. Cette phase consiste essentiellement à placer des *contrôles* sur la feuille et à en déterminer les positions respectives. Les contrôles sont des éléments constitutifs d'une fenêtre tels qu'une case à cocher, une zone de liste déroulante

ou un bouton de commande permettant une intervention de l'utilisateur. Un contrôle est un objet ; en tant que tel, il possède des propriétés, des méthodes et des événements définis.

Par exemple, un contrôle Checkbox (case à cocher) possède une propriété Value indiquant son état (True si la case est cochée, False si elle est décochée, et Null si elle n'est ni cochée ni décochée – grisée). Vous pouvez lui appliquer la méthode SetFocus pour lui affecter le focus, c'est-à-dire en faire l'objet recevant les événements clavier ou souris. Enfin, un événement Click (un clic de souris) affectant cet objet peut être détecté et entraîner un comportement spécifique de l'application, tel que le déclenchement d'une procédure (appelée procédure d'événement).

Certains contrôles, dits interactifs, réagissent aux actions de l'utilisateur (un bouton OK déclenchant la validation des données de la feuille et le déclenchement d'une procédure) ; les autres, dits statiques, ne sont accessibles qu'au niveau du code (un intitulé sur la feuille ne pouvant être modifié par l'utilisateur).

La Figure 4.15 présente la boîte de dialogue Police de Word sur laquelle vous pouvez visualiser différents types de contrôles.

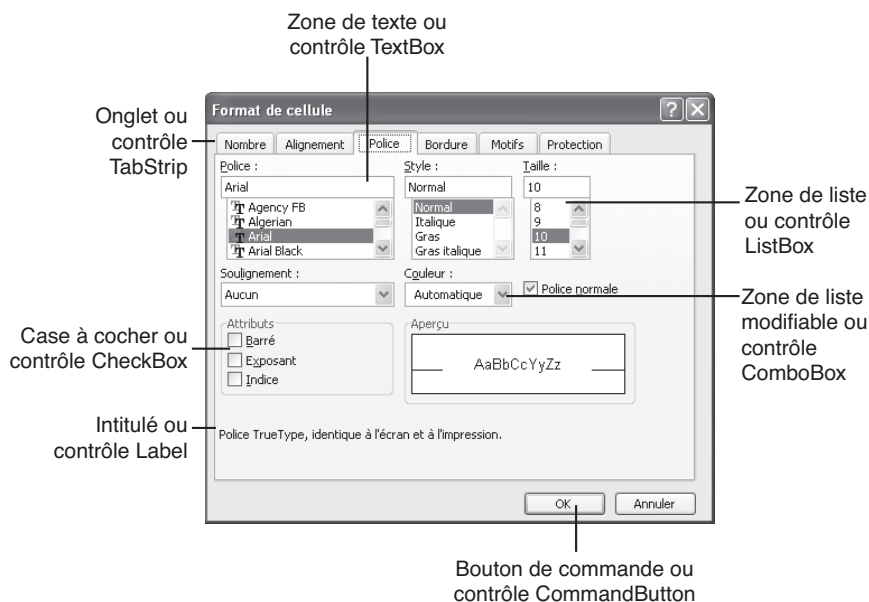


Figure 4.15

Les contrôles que vous pouvez placer sur une feuille sont les mêmes que ceux que vous rencontrez dans les applications Office.

Par défaut, la boîte à outils propose les contrôles les plus couramment rencontrés dans l'application hôte (voir Figure 4.16). Vous verrez au Chapitre 12 que vous pouvez personnaliser la boîte à outils en y ajoutant des contrôles ou en en modifiant l'ordonnancement.

Figure 4.16

Les contrôles de la boîte à outils peuvent très simplement être déposés sur une feuille.



Afficher et masquer la fenêtre UserForm et la boîte à outils

Pour afficher la fenêtre UserForm, vous devez soit ouvrir une feuille dans un projet (accessible dans le dossier Feuilles de l'Explorateur de projet), soit créer une nouvelle feuille. Pour quitter la fenêtre UserForm, cliquez sur la case de fermeture située dans l'angle supérieur droit de la fenêtre.

Pour afficher/masquer la boîte à outils, une fenêtre UserForm doit être active :

- Sélectionnez la commande Boîte à outils du menu Affichage.

ou :



- Cliquez sur le bouton Boîte à outils de la barre d'outils Standard de Visual Basic Editor.

Vous verrez au Chapitre 12 que Visual Basic Editor propose des outils facilitant grandement la phase de développement visuel des feuilles.

La fenêtre Code

La fenêtre Code est l'éditeur de code de Visual Basic Editor. Elle permet d'écrire du code Visual Basic ou de visualiser et de modifier le code existant. Toute action qu'exécute une

application VBA existe sous forme de code et l'ensemble du code constitutif d'un projet est édité dans la fenêtre Code.

Une fenêtre Code est toujours attachée à l'un des modules apparaissant dans l'Explorateur de projet (voir Figure 4.17). Il peut s'agir d'un module standard, d'un module de classe, d'une feuille ou du document auquel est attaché le projet.

La fenêtre Code présente de nombreuses et précieuses fonctionnalités, destinées à faciliter le développement de vos projets.

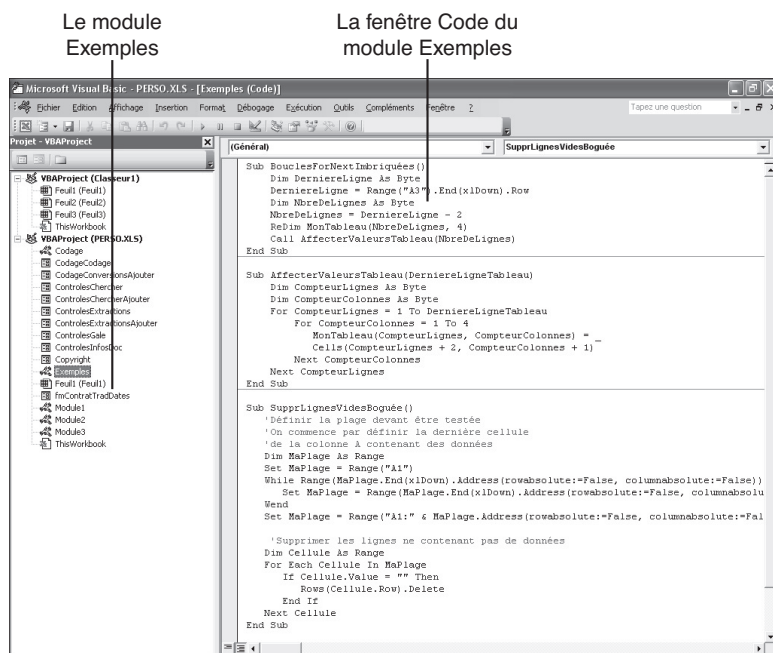
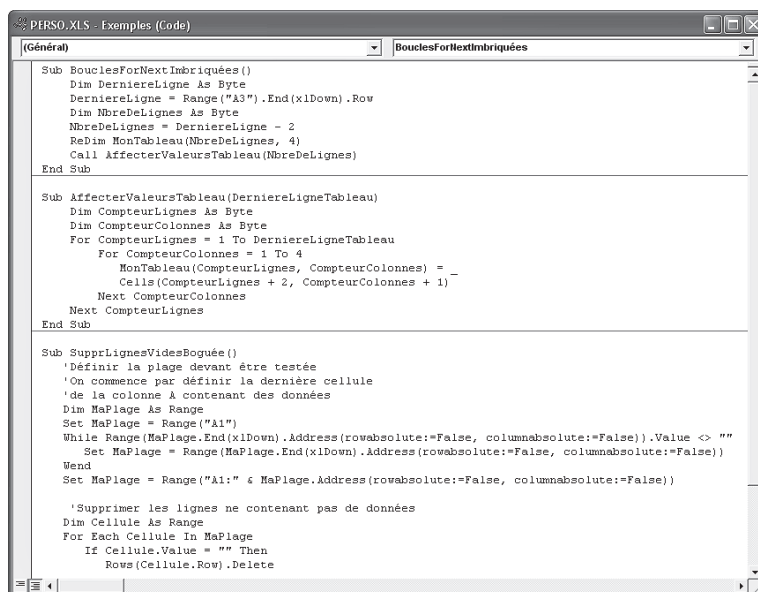


Figure 4.17
Chacun des modules d'un projet possède une fenêtre Code.

Vous pouvez, par exemple, ouvrir une fenêtre Code pour chacun des modules apparaissant dans votre projet et copier, coller ou déplacer du code d'une fenêtre et d'une procédure à l'autre, rechercher des chaînes de caractères ou marquer certains endroits du code en y plaçant des signets, paramétrer la fenêtre Code afin que la syntaxe Visual Basic soit vérifiée au fur et à mesure de la saisie du code, etc.

Figure 4.18
*La fenêtre Code
 est le cœur de
 Visual Basic Editor.*



Afficher et masquer la fenêtre Code

Pour accéder au code d'un élément constitutif d'un projet, ouvrez l'Explorateur de projet par l'une des méthodes indiquées ci-dessus et sélectionnez-y l'élément dont vous souhaitez afficher le code. Effectuez ensuite l'une des opérations suivantes :

- Double-cliquez sur un module de code.
- Cliquez du bouton droit sur l'élément voulu et choisissez la commande Code du menu contextuel.
- Cliquez sur le bouton Afficher le code de l'Explorateur de projet.
- Sélectionnez la commande Code du menu Affichage.
- Tapez le raccourci clavier F7.

La fenêtre Code de l'élément voulu s'affiche.

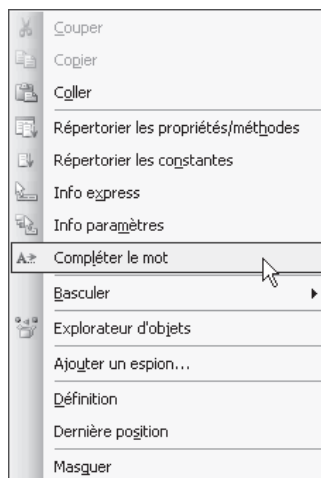


Pour accéder au code des contrôles d'une feuille, double-cliquez dessus ou cliquez du bouton droit et choisissez la commande Code du menu contextuel qui s'affiche.

Pour masquer la fenêtre Code, cliquez sur la case de fermeture ou cliquez du bouton droit dans la fenêtre et sélectionnez la commande Masquer du menu contextuel qui s'affiche.

Figure 4.19

Le menu contextuel de la fenêtre Code.



Naviguer dans la fenêtre Code

Par défaut, la fenêtre Code affiche l'ensemble des *procédures* constituant le module par ordre alphabétique, séparées par une ligne grise. Un même module pouvant contenir un nombre important de procédures, la fenêtre Code intègre deux zones de listes destinées à permettre un déplacement rapide dans le code du module affiché.



Une procédure est un bloc d'instructions, tel qu'une macro, délimité par des instructions d'encadrement et exécuté en tant qu'entité. Vous verrez au Chapitre 5 qu'une procédure peut avoir pour fonction de renvoyer une valeur ou d'exécuter un certain nombre d'actions. Le code exécutable des projets VBA est toujours contenu dans des procédures.

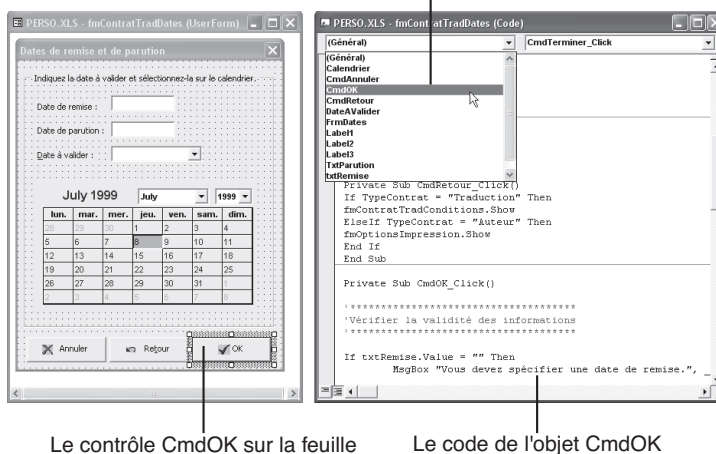
Les zones de listes Objet et Procédure

La zone de liste Objet référence tous les objets contenus dans le module (dans le cas d'une feuille, tous les contrôles). Pour accéder au code d'un objet, déroulez la zone de liste et sélectionnez-en le nom. Le code attaché à cet objet apparaît dans le haut de la fenêtre Code, le point d'insertion étant placé sur la première ligne du code en question (voir Figure 4.20).

Figure 4.20

La zone de liste
Objet répertorie l'ensemble
des objets du module.

L'objet CmdOK dans la zone de liste Objet



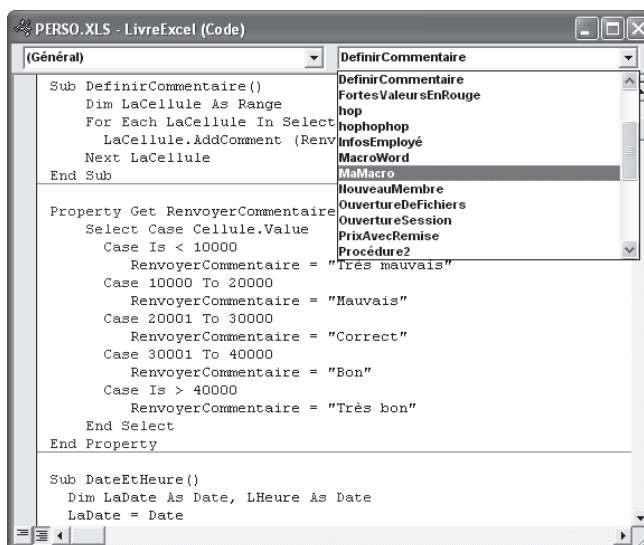
Le contrôle CmdOK sur la feuille

Le code de l'objet CmdOK

Les éléments référencés dans la zone de liste Procédure peuvent être des procédures ou des événements, selon le type du module. Si la fenêtre Code est celle d'un module de code, la zone de liste *Procédure* répertorie toutes les procédures du module en question. La Figure 4.21 présente la zone de liste Procédure dans la fenêtre Code du module LivreExcel. L'ensemble des procédures du projet y est référencé, ces dernières étant séparées par des lignes grises.

Figure 4.21

Utilisez cette zone pour
accéder rapidement à une
procédure du module.





Lorsque vous enregistrez ou créez des macros dans Excel, ces macros sont stockées dans un module nommé *Modulen*, accessible dans le dossier *Modules* de l'Explorateur de projet.

Dans le haut de la liste se trouve l'entrée (Déclarations). Il s'agit des *déclarations générales* du module. Si la fenêtre Code est celle d'une feuille, (Général) doit être sélectionné dans la zone Objet pour que (Déclarations) apparaisse dans la zone Procédure.

Si la fenêtre Code est celle d'une feuille, la zone de liste Procédure référence l'ensemble des événements (un clic de souris, une saisie au clavier, etc.) reconnus pour la feuille ou le contrôle sélectionné dans la zone de liste Objet (voir Figure 4.22) et la fenêtre Code affiche la procédure d'événement correspondante. Si l'option (Général) est sélectionnée dans la zone Objet, la zone Procédure répertorie l'ensemble des déclarations et des procédures de la feuille.

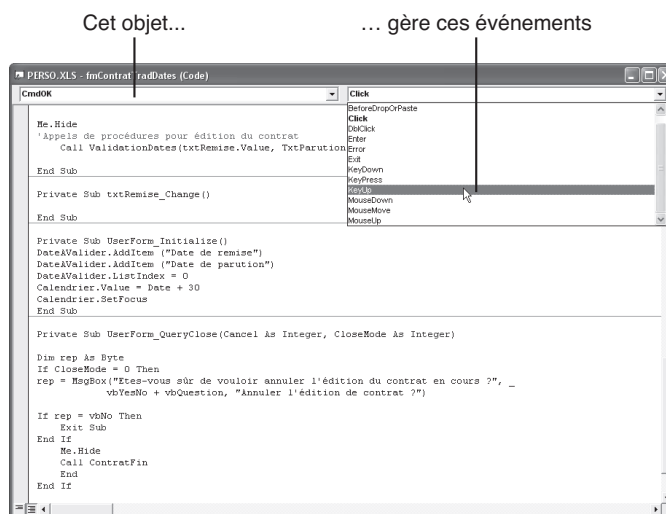


Une procédure d'événement (ou procédure événementielle) est une procédure attachée à un événement spécifique tel qu'un clic de souris. Ce type de procédure s'exécute lorsque l'événement est reconnu par le programme. La création de procédures d'événement est traitée au Chapitre 14.

Pour accéder à une procédure spécifique, déroulez la zone de liste et sélectionnez-en le nom. Le code de la procédure choisie apparaît dans le haut de la fenêtre Code, le point d'insertion étant automatiquement placé sur la première ligne de code de la procédure.

Figure 4.22

La zone de liste déroulante Procédure répertorie l'ensemble des événements référencés pour un objet.



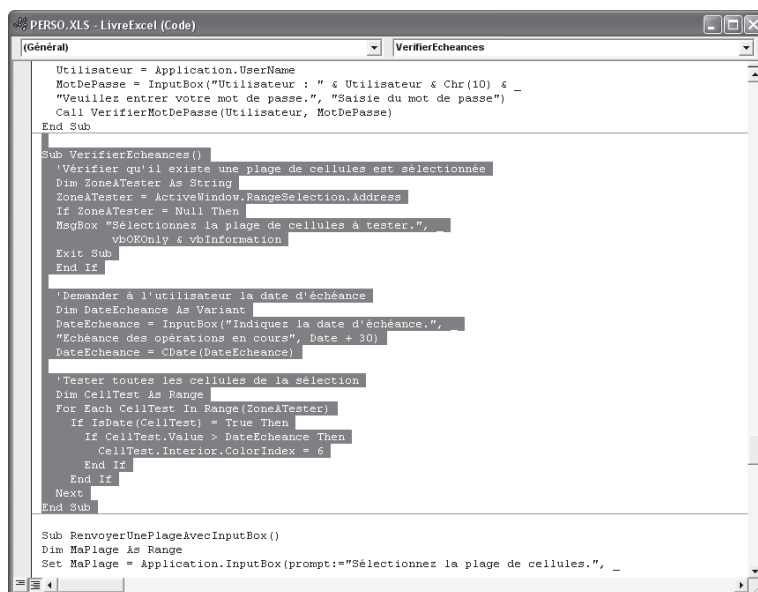
Sélection de texte dans la fenêtre Code

Lorsque vous vous trouvez dans une fenêtre Code, vous pouvez sélectionner du texte à l'aide de la souris ou du clavier. Nous supposons ici que vous connaissez les méthodes de sélection standard à l'aide de ces deux périphériques. Il existe cependant des modes de sélection spécifiques à la fenêtre Code de Visual Basic Editor.

- Pour sélectionner une procédure à l'aide de la souris, placez le pointeur sur la gauche du code de façon qu'il prenne la forme d'une flèche orientée vers la droite et double-cliquez (voir Figure 4.23).

Figure 4.23

Vous pouvez sélectionner rapidement une procédure entière.



- Vous pouvez aussi vous déplacer ou effectuer des sélections de blocs de code à l'aide du clavier. Pour atteindre la procédure précédente, faites le raccourci clavier Ctrl+Pg. Préc. Pour atteindre la procédure suivante, faites le raccourci Ctrl+Pg. Suiv. Pour étendre la sélection, utilisez les mêmes combinaisons de touches auxquelles vous ajouterez la touche Maj.

Astuce

La substitution des touches fléchées haut et bas aux touches Pg. Préc. et Pg. Suiv. dans les combinaisons mentionnées ci-dessus permet d'atteindre la première ligne de code sous la déclaration de procédure (Sub, Function ou Property) plutôt que la déclaration de procédure elle-même.

Recherche et remplacement de texte

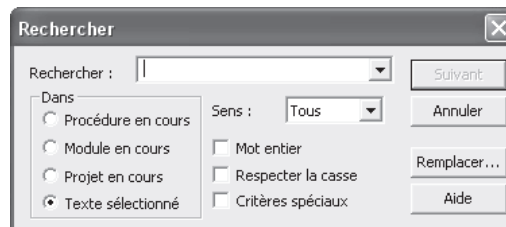
Vous pouvez rechercher du texte dans la fenêtre Code comme vous le feriez dans un logiciel de traitement de texte tel que Word. Pour rechercher du texte dans la fenêtre Code, procédez comme suit :

1. À partir d'une fenêtre Code, sélectionnez la commande Rechercher du menu Édition ou tapez le raccourci clavier Ctrl+F, ou cliquez sur le bouton Rechercher de la barre d'outils Standard de Visual Basic Editor.

La boîte de dialogue Rechercher représentée à la Figure 4.24 s'affiche.

Figure 4.24

La boîte de dialogue Rechercher de Visual Basic Editor propose des options communes à l'essentiel des traitements de texte.



2. Dans la zone de texte Rechercher, saisissez le texte voulu.
3. Dans la zone Dans, sélectionnez la portée de la recherche :
 - **Procédure en cours.** La recherche ne porte que sur le texte de la procédure en cours, c'est-à-dire celle dans laquelle se trouve le curseur.
 - **Module en cours.** La recherche porte sur l'ensemble du module en cours, c'est-à-dire le texte de la fenêtre Code active.
 - **Projet en cours.** La recherche porte sur l'ensemble des modules du projet, que leurs fenêtres de codes respectives soient ouvertes ou non.
 - **Texte sélectionné.** La recherche porte sur la plage de texte sélectionnée dans la fenêtre active.
4. Dans la zone de liste déroulante, sélectionnez le sens dans lequel s'effectuera la recherche par rapport à l'emplacement du curseur : vers le haut, vers le bas ou dans les deux sens.
5. Cochez éventuellement les cases des options de recherche :
 - **Mot entier.** Le mot est recherché en tant que mot entier, et non en tant que suite de caractères faisant partie d'un autre mot.
 - **Respecter la casse.** La recherche porte sur le texte dont la casse (majuscules ou minuscules) est identique à celle du texte saisi dans la zone Rechercher.
 - **Critères spéciaux.** Lorsque cette case est cochée, vous pouvez utiliser les caractères génériques (?, *, #, [listedecar], et [!listedecar]) dans la zone de texte Rechercher.

6. Cliquez sur le bouton Suivant pour lancer la recherche.

Si la recherche aboutit, le texte trouvé s'affiche en surbrillance dans la fenêtre Code. (Si la recherche porte sur le projet entier et si la chaîne est trouvée dans un autre module, la fenêtre de Code de ce module est ouverte.) Pour rechercher une autre occurrence du texte, cliquez de nouveau sur le bouton Suivant.

Si le texte recherché n'est pas trouvé, un message vous l'indique.

Pour fermer la boîte de dialogue Rechercher, cliquez sur le bouton Annuler ou sur sa case de fermeture.



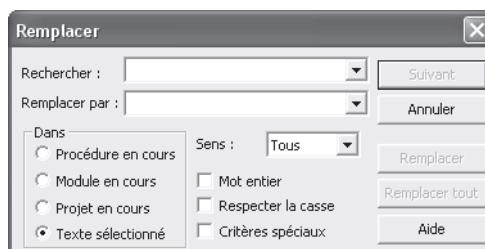
Une pression sur la Touche F3 ou la commande Suivant du menu Édition relance la dernière recherche effectuée sans ouvrir la boîte de dialogue Rechercher.

Pour remplacer du texte, procédez comme suit :

1. Sélectionnez la commande Remplacer du menu Édition ou tapez le raccourci clavier Ctrl+H. Si la boîte de dialogue Rechercher est ouverte, cliquez sur le bouton Remplacer. La boîte de dialogue Remplacer s'affiche (voir Figure 4.25).

Figure 4.25

La boîte de dialogue Remplacer.



2. Indiquez le texte à rechercher et définissez l'étendue, le sens et les options de recherche. Dans la zone Remplacer par, saisissez le texte de remplacement.
3. Si vous souhaitez effectuer le remplacement sur toutes les occurrences du texte recherché dans la zone de recherche définie, cliquez sur Remplacer tout.

Une boîte de dialogue vous indique le nombre de remplacements effectués.

4. Pour visualiser les occurrences du texte trouvé avant d'effectuer le remplacement, cliquez sur le bouton Suivant pour lancer la recherche. Lorsque le texte est trouvé, il apparaît en surbrillance dans la fenêtre Code.

Pour remplacer l'occurrence sélectionnée, cliquez sur le bouton Remplacer. Le texte est remplacé et la recherche se poursuit.

Pour poursuivre la recherche sans effectuer de remplacement, cliquez sur le bouton Suivant.

Affichage de plusieurs fenêtres Code

Lorsque vous développerez dans Visual Basic Editor, vous serez souvent amené à vous déplacer entre les fenêtres Code de différents éléments de votre projet, voire à échanger du code d'une fenêtre à l'autre pour en épargner une nouvelle saisie. Pour passer d'une fenêtre Code à l'autre, ouvrez le menu Fenêtre de la barre de menus de Visual Basic Editor et sélectionnez la fenêtre que vous souhaitez passer au premier plan.

Pour afficher simultanément plusieurs fenêtres Code, procédez comme suit :

1. Placez-vous dans l'Explorateur de projet et ouvrez les fenêtres de code concernées selon la procédure décrite précédemment dans ce chapitre.
2. Sélectionnez les commandes Mosaïque horizontale (voir Figure 4.26) ou Mosaïque verticale (voir Figure 4.27) du menu Fenêtre, de façon à visualiser simultanément toutes les fenêtres Code ouvertes.

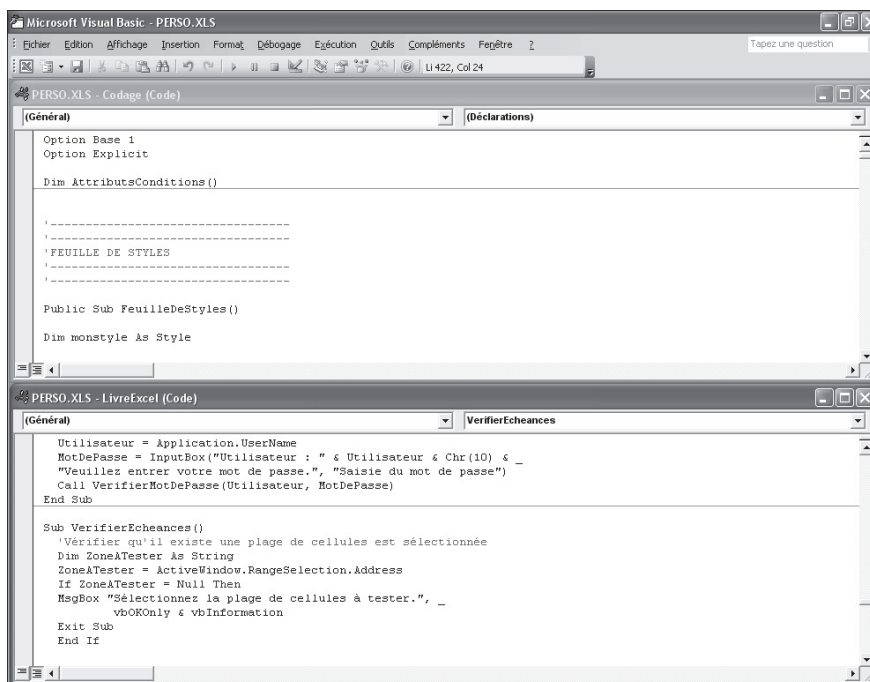
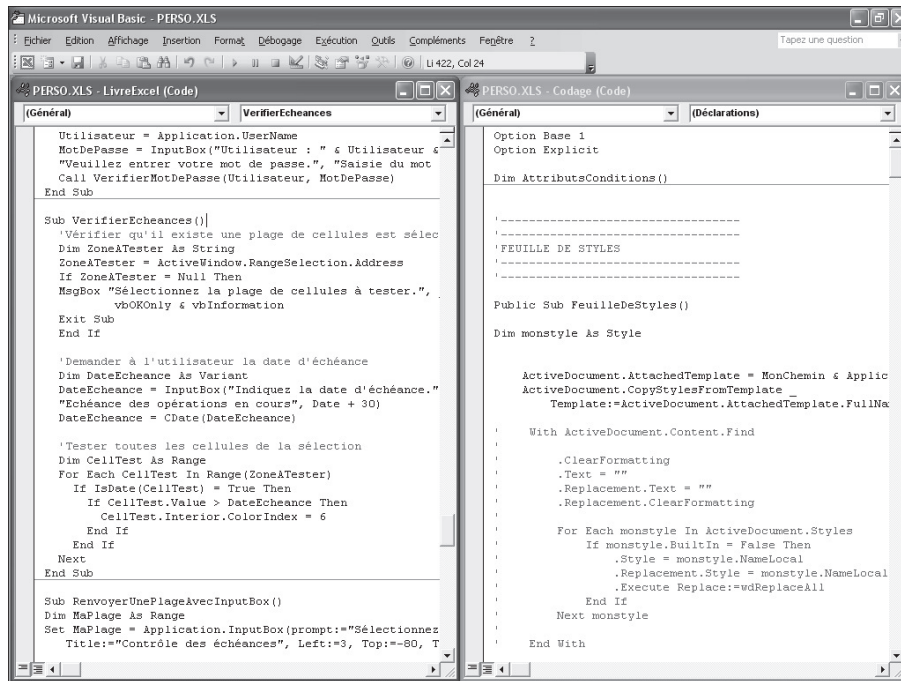


Figure 4.26

Les fenêtres Code organisées en mosaïque horizontale.

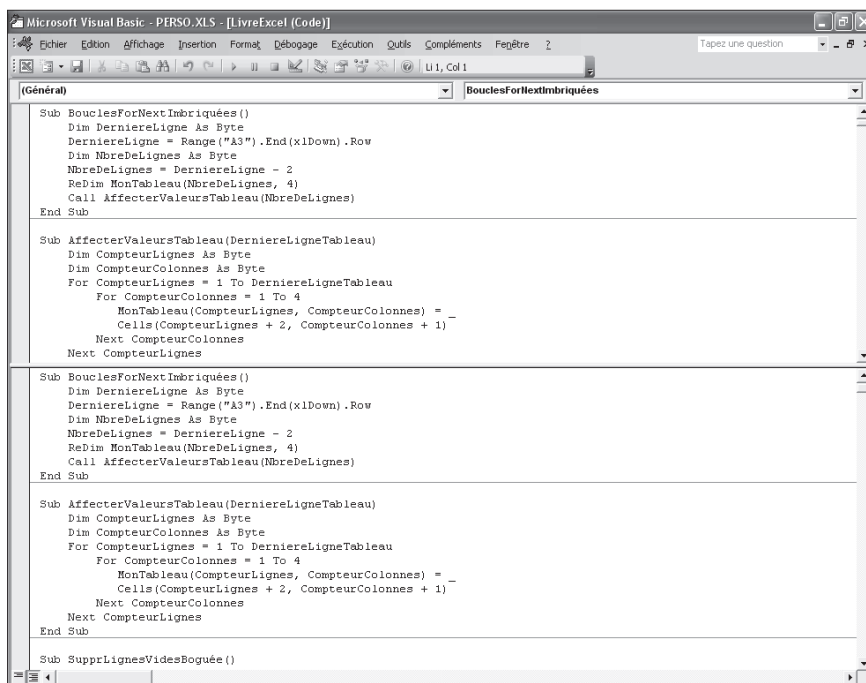
**Figure 4.27**

Les fenêtres Code organisées en mosaïque verticale.

Vous pouvez aussi séparer une fenêtre Code en deux volets, afin d'afficher simultanément des sections de code non contiguës d'un même module. Vous pouvez ainsi copier, coller ou déplacer du code d'une section à l'autre du module. Procédez comme suit :

1. Placez le curseur sur la barre de fractionnement située dans le haut de la barre de défilement verticale de la fenêtre Code.
2. Lorsque le curseur se transforme en une double flèche, faites glisser la barre de fractionnement vers le bas, jusqu'à atteindre la délimitation souhaitée pour les deux volets. Relâchez le bouton de la souris.

La fenêtre Code étant séparée en deux volets autonomes (voir Figure 4.28), vous pouvez modifier l'affichage de l'un ou l'autre des volets à l'aide des barres de défilement verticales. Les zones de listes Objet et Procédure s'appliquent à la fenêtre active, c'est-à-dire à celle dans laquelle se trouve le point d'insertion.

**Figure 4.28**

Le fractionnement de la fenêtre Code permet d'afficher simultanément différentes sections de code d'un même module.

Pour annuler le fractionnement de la fenêtre Code, double-cliquez sur la barre de fractionnement ou faites-la glisser vers le haut de la fenêtre.

Options d'affichage

L'affichage du code dans la fenêtre Code de Visual Basic Editor peut être paramétré de différentes façons. Cette section présente les options de paramétrage de la fenêtre. Les options de paramétrage du code sont présentées au Chapitre 5.

Par défaut, la fenêtre Code affiche l'ensemble des procédures du module, chaque procédure étant séparée par un trait gris (voir Figure 4.29).

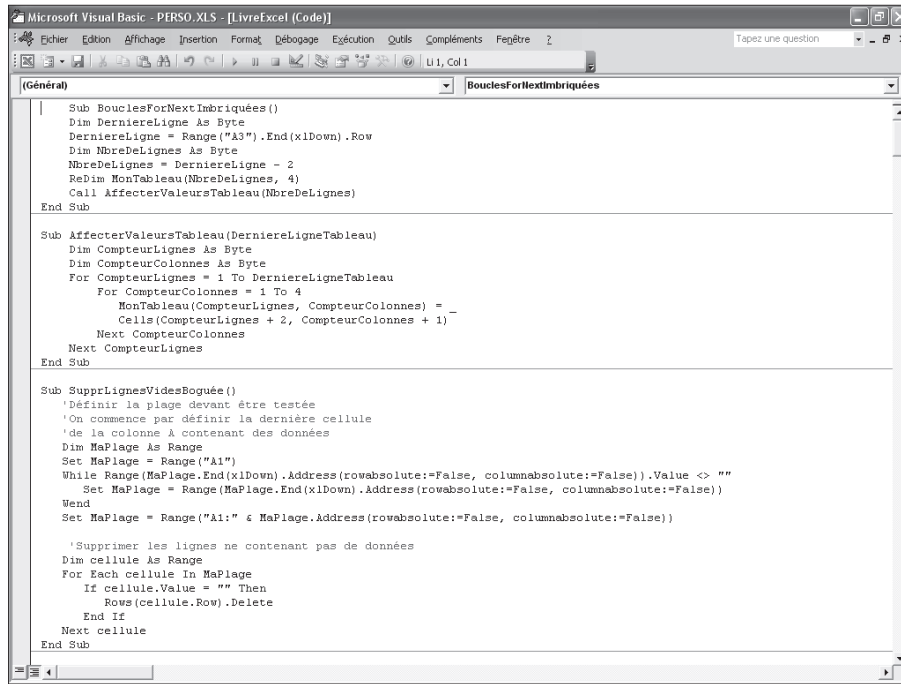


Figure 4.29

L'ensemble des procédures du code s'affiche.

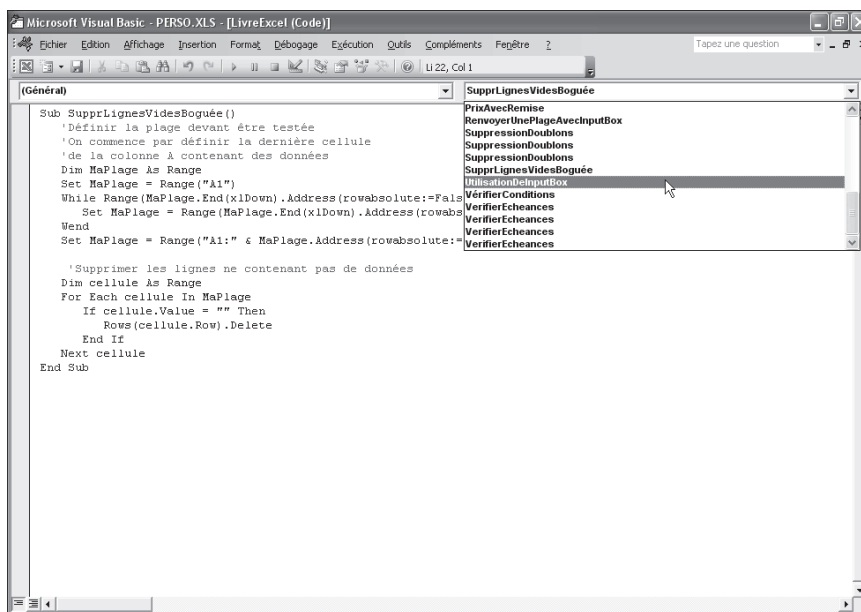
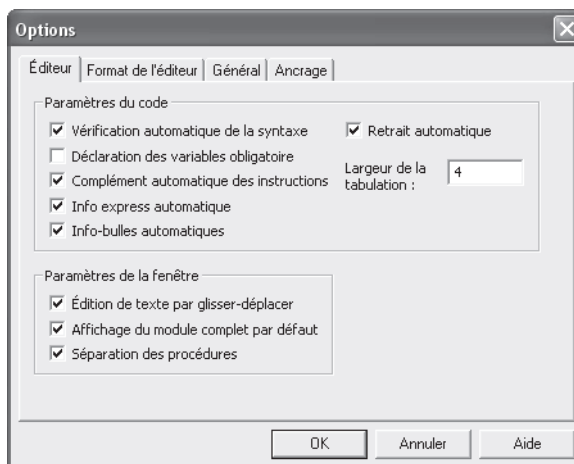
Si l'affichage de l'ensemble des procédures du module ne vous convient pas, vous pouvez choisir de n'afficher qu'une procédure dans la fenêtre Code. Pour définir les paramètres de la fenêtre Code, procédez comme suit :

1. Choisissez la commande Options du menu Outils. Sélectionnez l'onglet Éditeur de la boîte de dialogue Options qui s'affiche (voir Figure 4.30).
2. Cochez ou décochez les cases de la zone Paramètres de la fenêtre, en fonction du type d'affichage souhaité :
 - **Glisser-déplacer pour l'édition de texte.** Cochez cette case pour autoriser les glisser-déplacer de texte dans la fenêtre Code.
 - **Affichage complet du module par défaut.** Décochez cette case si vous souhaitez n'afficher qu'une procédure dans la fenêtre Code (voir Figure 4.31). Utilisez alors la zone de liste déroulante Procédure pour sélectionner la procédure que vous souhaitez afficher.

- **Séparation des procédures.** Dans le cas d'un affichage complet du module, cette option détermine si un séparateur sera affiché ou non entre les différentes procédures affichées.
3. Cliquez ensuite sur OK pour valider les options choisies.

Figure 4.30

La zone Paramètres de la fenêtre permet de définir l'affichage des procédures dans la fenêtre Code.

**Figure 4.31**

Vous pouvez choisir de n'afficher qu'une procédure dans la fenêtre Code.

La fenêtre Propriétés

La fenêtre Propriétés permet d'accéder aux propriétés de divers éléments d'un projet. Vous pouvez y visualiser et modifier les propriétés d'un classeur ou d'une feuille de calcul, d'une feuille UserForm, d'un contrôle, d'une classe, d'un projet ou d'un module. Vous utiliserez la fenêtre Propriétés essentiellement lors du développement de feuilles pour vos projets.



Vous découvrirez les propriétés essentielles des contrôles ActiveX que l'on place sur les feuilles UserForm et apprendrez à les modifier aux Chapitres 13 et 15.

Les propriétés varient en fonction de l'élément sélectionné. Ainsi les propriétés d'un contrôle en définissent l'aspect et le comportement (sa position, sa taille, le fait que l'utilisateur peut ou non y apporter des modifications, etc.), tandis que les propriétés d'un projet en définissent simplement le nom. La Figure 4.32 présente la fenêtre Propriétés d'un contrôle CheckBox (case à cocher) et la Figure 4.33, les propriétés du projet PERSONAL.XLSB.

Figure 4.32

La fenêtre Propriétés d'un contrôle CheckBox.

La colonne gauche affiche les noms des propriétés

La colonne droite affiche les valeurs affectées aux propriétés

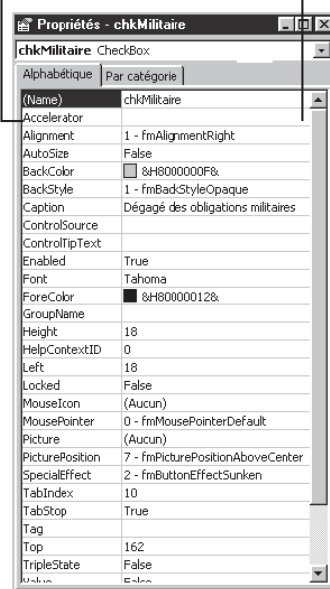
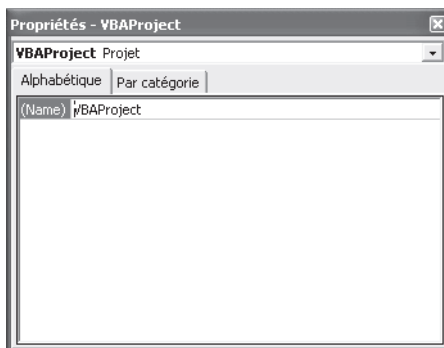


Figure 4.33

La fenêtre Propriétés du projet PERSONAL.XLSB.



Afficher et masquer la fenêtre Propriétés

La fenêtre Propriétés peut être affichée à partir de l'Explorateur de projet ou à partir d'une fenêtre UserForm. Pour afficher une fenêtre Propriétés, procédez comme suit :

1. Dans l'Explorateur de projet, sélectionnez le module dont vous souhaitez afficher les propriétés ou, dans la fenêtre UserForm, sélectionnez le contrôle ou la feuille dont vous souhaitez afficher les propriétés.



Lorsque plusieurs contrôles sont sélectionnés sur une feuille, la fenêtre Propriétés affiche les propriétés communes aux contrôles sélectionnés.



2. Dans le menu Affichage sélectionnez la commande Fenêtre Propriétés, ou cliquez sur le bouton Propriétés de la barre d'outils Standard, ou encore tapez le raccourci clavier F4.

La fenêtre Propriétés du module sélectionné s'affiche.

3. Lorsque vous sélectionnez un autre module dans l'Explorateur de projet ou un autre contrôle dans la fenêtre UserForm, la fenêtre Propriétés affiche les propriétés de l'élément sélectionné.

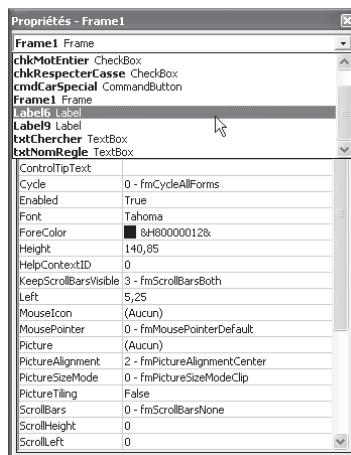
Pour masquer la fenêtre Propriétés, cliquez sur la case de fermeture située à l'extrémité droite de la barre de titre ou cliquez du bouton droit dans la fenêtre et, dans le menu contextuel qui s'affiche, sélectionnez la commande Masquer.

Naviguer dans la fenêtre Propriétés

Lorsque vous vous trouvez dans la fenêtre Propriétés d'une feuille ou d'un contrôle, une zone de liste déroulante permet d'accéder aux propriétés des contrôles de la feuille active. Déroulez simplement la liste et sélectionnez le contrôle dont vous souhaitez afficher les propriétés (voir Figure 4.34).

Figure 4.34

Sélectionnez l'objet dont vous souhaitez afficher les propriétés.



La fenêtre Propriétés présente deux onglets déterminant le type d'affichage des propriétés :

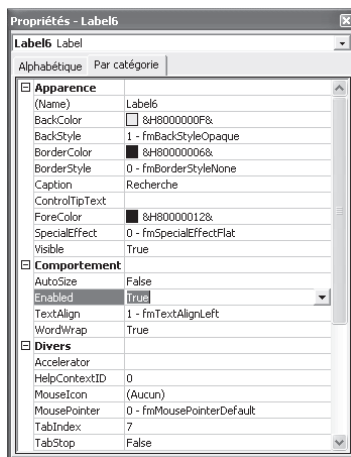
- **Alphabétique.** Toutes les propriétés recensées pour l'élément sélectionné sont affichées par ordre alphabétique (voir Figure 4.32).
- **Par catégorie.** Les propriétés de l'objet sélectionné sont regroupées par catégorie et, à l'intérieur de chaque catégorie, par ordre alphabétique (voir Figure 4.35). Les catégories apparaissent en gras. À l'instar des dossiers de l'Explorateur, les propriétés qui y sont recensées peuvent être affichées ou masquées lorsque vous cliquez sur les signes plus (+) ou moins (–) situés à gauche de chaque nom de catégorie.

Les catégories de propriétés essentielles sont :

- **Apparence.** Sous cette catégorie sont regroupées les propriétés relatives à l'apparence d'un objet (sa couleur de fond, son intitulé, le texte de son info-bulle, s'il est ou non visible, etc.).
- **Comportement.** Cette catégorie regroupe les propriétés déterminant le comportement du contrôle, c'est-à-dire ses réactions aux actions de l'utilisateur (s'il est ou non modifiable, longueur maximale de saisie d'une chaîne, etc.).

Figure 4.35

L'affichage par catégorie permet d'accéder rapidement aux propriétés voulues.



- **Défilement.** Cette catégorie regroupe les propriétés concernant le déplacement dans un contrôle (présence ou non de barres de défilement, dimension et position relative des barres de défilement).
- **Divers.** Vous trouverez sous cette catégorie des propriétés variées, telles que le nom permettant d'appeler ce contrôle dans le code.
- **Emplacement.** Les propriétés concernant la taille d'un contrôle et sa position sur la feuille sont regroupées sous cette catégorie.
- **Image.** Si une image est associée à un contrôle, vous trouverez sous cette catégorie les propriétés spécifiques à cette image (source, position relative).
- **Police.** La police utilisée pour l'intitulé du contrôle s'affiche sous cette catégorie. Dans Office, la police utilisée par défaut pour les contrôles est le Tahoma.

Modifier une propriété

Comme nous l'avons vu au Chapitre 1, différents types de valeurs peuvent être affectées à une propriété. Il peut s'agir :

- d'une chaîne de caractères ;
- d'une valeur numérique ;
- d'une constante ;
- d'une valeur booléenne.

Pour modifier la valeur d'une propriété dans la fenêtre Propriétés, procédez comme suit :

1. Sélectionnez cette propriété dans la fenêtre Propriétés. Elle apparaît en surbrillance. Appuyez sur la touche Tab pour sélectionner la valeur en cours pour cette propriété.
2. Affectez ensuite la valeur voulue à la propriété. La démarche varie selon le type de valeur affecté à la propriété :
 - Si la propriété accepte une chaîne de caractères ou une valeur numérique, saisissez directement la valeur souhaitée au clavier.



- Si le type de valeur affecté à cette propriété est une valeur booléenne ou une constante, une flèche apparaît dans la cellule de valeur. cliquez sur cette flèche pour dérouler la liste des valeurs possibles pour la propriété, et sélectionnez celle qui convient (voir Figure 4.36).

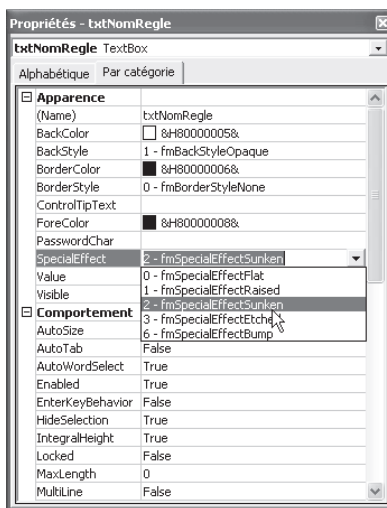
Vous pouvez aussi utiliser les touches fléchées haut et bas pour vous déplacer d'une valeur à l'autre.



- Si la propriété attend pour valeur une chaîne indiquant un emplacement de fichier, un bouton permettant de parcourir les fichiers disponibles à la recherche du fichier voulu apparaît dans la zone de valeur de la propriété.

Figure 4.36

Lorsqu'une propriété accepte des valeurs prédéterminées, celles-ci sont recensées dans une liste déroulante.



Astuce

Dans le cas des propriétés acceptant une valeur booléenne ou une constante, vous pouvez passer d'une valeur à la suivante en double-cliquant simplement dans la case de valeur d'une propriété.

Les barres d'outils

Visual Basic Editor contient quatre barres d'outils, présentées dans les figures suivantes.

Figure 4.37

La barre d'outils Standard permet d'accéder aux fonctions les plus communes de Visual Basic Editor.



Figure 4.38

La barre d'outils Édition permet d'obtenir de l'aide, de mettre en forme le texte et de s'y déplacer.



Figure 4.39

La barre d'outils Débogage permet de tester le comportement d'un programme.

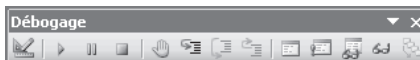


Figure 4.40

La barre d'outils UserForm propose des outils pour organiser les contrôles sur une feuille.



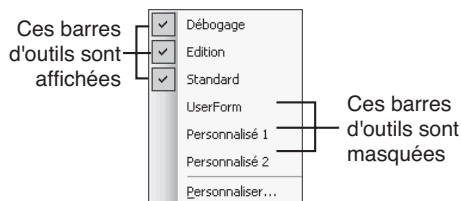
Cette section présente la barre d'outils Standard. Vous découvrirez les autres barres d'outils de Visual Basic Editor au fur et à mesure de la lecture de cet ouvrage.

Afficher, masquer et déplacer une barre d'outils

Pour afficher ou masquer une barre d'outils, affichez le menu contextuel Barre d'outils, soit en cliquant du bouton droit sur une barre d'outils ou sur la barre de menus de la fenêtre Visual Basic Editor, soit en sélectionnant la commande Barre d'outils du menu Affichage. Dans ce menu, sélectionnez la barre d'outils que vous souhaitez afficher ou masquer – une coche à côté du nom d'une barre d'outils indiquant qu'elle est déjà affichée.

Figure 4.41

Le menu contextuel Barre d'outils.



Pour déplacer une barre d'outils, cliquez sur celle-ci et, tout en en maintenant le bouton enfoncé, déplacez la souris jusqu'à atteindre l'emplacement voulu. Un contour grisé vous indique l'emplacement que prendra la barre d'outils. Relâchez le bouton de la souris lorsque le contour indique l'emplacement voulu.



Pour faire apparaître les info-bulles associées aux boutons de la barre d'outils, sélectionnez la commande Options du menu Outils et, dans l'onglet Général, cochez l'option Afficher les info-bulles.

La barre d'outils Standard

La barre d'outils Standard offre un accès rapide aux commandes les plus usitées. On y retrouve des commandes communes à l'essentiel des applications, telles que Enregistrer ou Couper. Le Tableau 4.2 présente les boutons de cette barre d'outils.

Tableau 4.2 : Les icônes de la barre d'outils Standard

<i>Bouton</i>	<i>Nom</i>	<i>Description</i>
	Afficher Microsoft Excel	Bascule entre l'application hôte et le document Visual Basic actif (raccourci clavier : Alt+F11).
	Insertion	En cliquant sur la flèche, vous ouvrez un menu permettant d'insérer l'un des objets suivants dans le projet actif :  UserForm (feuille)  Module de classe  Module  Procédure En cliquant sur le bouton, vous insérez l'objet représenté. Le bouton représente le dernier objet ajouté. Le bouton par défaut est la feuille (UserForm).
	Enregistrer <Document hôte>	Enregistre le document hôte, y compris le projet et tous ses composants – feuilles et modules (raccourci clavier : Ctrl+S).

<i>Bouton</i>	<i>Nom</i>	<i>Description</i>
	Couper	Supprime le texte sélectionné dans la fenêtre Code ou le contrôle sélectionné sur la feuille active et le place dans le Presse-papiers (raccourci clavier : Ctrl+X).
	Copier	Copie le texte sélectionné dans la fenêtre Code ou le contrôle sélectionné sur la feuille active dans le Presse-papiers (raccourci clavier : Ctrl+C).
	Coller	Insère le contenu du Presse-papiers à l'emplacement courant. Ce bouton n'est accessible que si le contenu du Presse-papiers est compatible avec la fenêtre active – fenêtre Code pour du texte et fenêtre UserForm pour un contrôle (raccourci clavier : Ctrl+V).
	Rechercher	Permet de rechercher une chaîne dans une fenêtre Code – voir la section "La fenêtre Code", plus haut dans ce chapitre (raccourci clavier : Ctrl+F).
	Annuler	Chaque clic sur ce bouton annule la dernière modification effectuée (raccourci clavier : Ctrl+Z).
	Répéter	Chaque clic sur ce bouton rétablit la dernière action annulée. Pour qu'il soit accessible, il faut qu'une action ait été annulée et qu'aucune modification n'ait eu lieu depuis l'annulation.
	Exécuter Sub/UserForm ou Exécuter la macro	Exécute une application fonction de la fenêtre active : • Fenêtre Code : la procédure en cours est exécutée. • Fenêtre UserForm : la feuille active est exécutée. • Autre : ouvre la boîte de dialogue Macros du projet actif, à partir de laquelle vous pouvez exécuter la macro de votre choix.
	Arrêt	Interrompt l'exécution en cours et bascule en mode Arrêt. En mode Arrêt, vous pouvez relancer l'exécution d'une procédure, réinitialiser un projet, tester un programme, etc. (raccourci clavier : Ctrl+Pause).
	Réinitialiser	Réinitialise le projet.
	Mode Création/Quitter le mode Création	Active ou désactive le mode Création. On parle de mode Création par opposition au mode Exécution. En mode Création, le code d'un projet n'est pas exécuté. Cela correspond à la période de développement de votre projet – création d'une feuille UserForm ou écriture de code. En mode Création, vous pouvez placer des contrôles ActiveX sur les feuilles de calcul et leur affecter du code.

Tableau 4.2 : Les icônes de la barre d'outils Standard (*suite*)

<i>Bouton</i>	<i>Nom</i>	<i>Description</i>
	Explorateur de projet	Affiche la fenêtre Explorateur de projet (raccourci clavier : Ctrl+R).
	Fenêtre Propriétés	Affiche la fenêtre Propriétés de l'objet sélectionné (raccourci clavier : F4).
	Explorateur d'objets	Affiche la fenêtre Explorateur d'objets (raccourci clavier : F2).
	Boîte à outils	Affiche la boîte à outils. Accessible uniquement si une fenêtre UserForm est active.
	Assistant Office	Ouvre la fenêtre de l'Assistant Office, dans laquelle vous êtes invité à saisir le sujet sur lequel vous souhaitez obtenir de l'aide (raccourci clavier : F1).

Paramétrer Visual Basic Editor

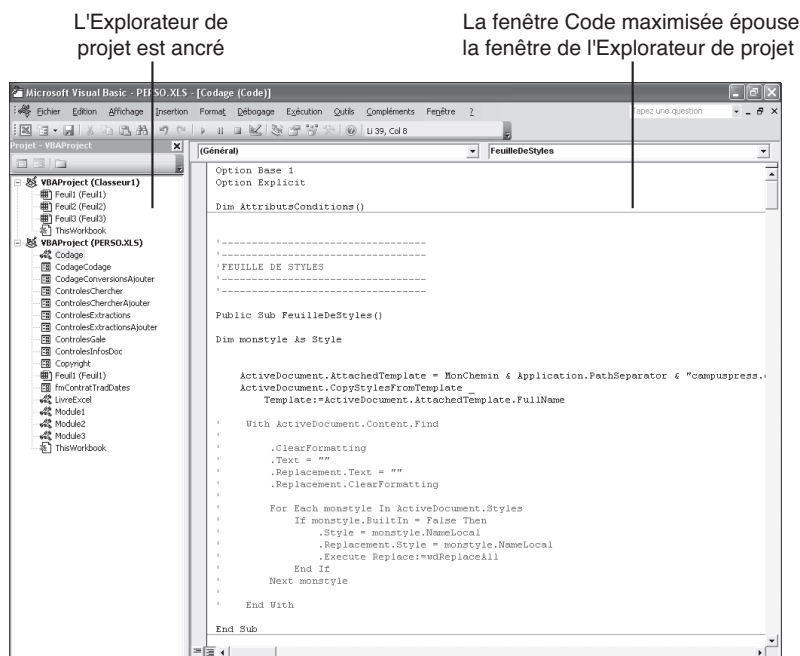
L'affichage de Visual Basic Editor peut être paramétré et personnalisé. Testez ces options de façon à trouver les paramétrages qui vous conviennent le mieux.

Comme la plupart des fenêtres Windows, les fenêtres de Visual Basic Editor peuvent être déplacées en opérant un cliquer-glisser sur leur barre de titre, et redimensionnées en en tirant les bords. Vous pouvez aussi choisir d'ancrer une ou plusieurs fenêtres à l'une des bordures de la fenêtre de Visual Basic Editor.



Une fenêtre est dite ancrée lorsqu'elle est fixée à la bordure d'une autre fenêtre. Une fenêtre peut être ancrée à la fenêtre de l'application ou à une autre fenêtre elle-même ancrée.

Une fenêtre ancrée demeure au premier plan, et ce même lorsqu'une autre fenêtre dont l'espace recouvre le sien est activée. Par ailleurs, si vous maximisez une fenêtre dans Visual Basic Editor, celle-ci viendra épouser la fenêtre ancrée, sans en recouvrir l'espace. Vous pouvez ainsi afficher plusieurs fenêtres, tout en tirant pleinement parti de l'espace de Visual Basic Editor. À la Figure 4.42, nous avons ancré l'Explorateur de projet et maximisé une fenêtre Code.

**Figure 4.42**

Ancrer l'Explorateur de projet permet d'accéder rapidement aux éléments de vos projets.

Les fenêtres pouvant être ancrées sont :

- la fenêtre Exécution ;
- la fenêtre Variables locales ;
- la fenêtre Espions ;
- la fenêtre Explorateur de projet ;
- la fenêtre Propriétés ;
- la fenêtre Explorateur d'objets.

Lorsqu'une fenêtre est ancrée, elle se place automatiquement sur l'une des bordures de la fenêtre de Visual Basic Editor quand vous la déplacez. Si vous souhaitez pouvoir la déplacer n'importe où dans la fenêtre de Visual Basic Editor, désactivez-en l'ancrage.

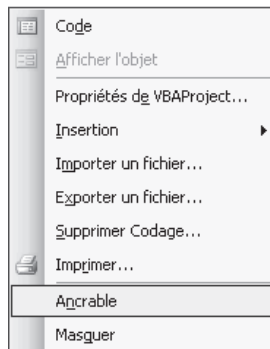
Pour activer ou désactiver l'ancrage des fenêtres de Visual Basic Editor, choisissez l'une des méthodes suivantes :

- Cliquez du bouton droit dans la fenêtre de votre choix. Un menu contextuel s'affiche, en fonction de la fenêtre choisie. Ces menus contextuels ont en commun de posséder

une commande Ancrable (voir Figure 4.43). La sélection de cette commande active ou désactive l'ancrage de la fenêtre, une coche indiquant que la fenêtre est ancrée.

Figure 4.43

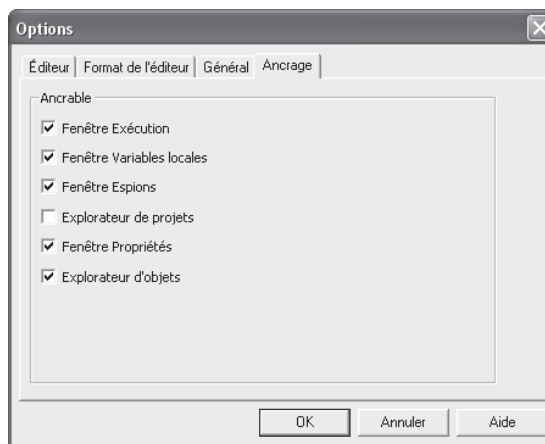
Les menus contextuels des fenêtres de Visual Basic Editor possèdent une commande Ancrable.



- Sélectionnez la commande Options du menu Outils et placez-vous sur l'onglet Ancrage (voir Figure 4.44). Cochez les cases des fenêtres que vous souhaitez ancrer et décochez les cases des fenêtres dont vous voulez désactiver l'ancrage. Cliquez sur OK.

Figure 4.44

La boîte de dialogue Options permet de modifier les options d'ancrage de plusieurs fenêtres simultanément.



Info

Si l'accès aux commandes de Visual Basic Editor ne convient pas à votre façon de travailler, vous pouvez choisir d'en personnaliser les menus et les barres d'utils. Vous pouvez ainsi ajouter ou supprimer des commandes aux menus et des boutons sur les barres d'utils. Vous pouvez aussi créer une nouvelle barre d'utils ou un nouveau menu dans lesquels vous placerez les commandes de votre choix. Pour connaître les procédures de personnalisation de menus et de barres d'utils, reportez-vous au Chapitre 11.



Programmer en Visual Basic

CHAPITRE 5. Développer dans Visual Basic Editor

CHAPITRE 6. Variables et constantes

CHAPITRE 7. Contrôler les programmes VBA

CHAPITRE 8. Fonctions Excel et VBA

CHAPITRE 9. Manipulation des chaînes de caractères

CHAPITRE 10. Débogage et gestion des erreurs

CHAPITRE 11. Intégrer des applications VBA dans l'interface d'Excel



5

Développer dans Visual Basic Editor

Au sommaire de ce chapitre

- Structure des programmes Visual Basic
- Les différents types de procédures
- Des projets bien structurés
- Créer une procédure
- Appel et sortie d'une procédure
- Exécuter du code
- Aide à l'écriture de code

Les chapitres précédents vous ont permis d'acquérir les concepts essentiels de la programmation Visual Basic pour Applications et de découvrir l'environnement de développement Visual Basic Editor. Avec ce chapitre, nous entrons de plain-pied dans la programmation VBA.

Vous apprendrez à distinguer les composants essentiels d'un projet et à déterminer les besoins de votre projet. Vous serez ainsi à même de structurer de façon cohérente vos projets, leur assurant efficacité et lisibilité. Gardez à l'esprit que les projets sont attachés à une application hôte (ici Excel). Pour créer un nouveau projet ou accéder à un projet existant, vous devez donc ouvrir Visual Basic Editor à partir d'Excel, selon les procédures décrites aux chapitres précédents. Le classeur auquel est affecté (ou auquel vous souhaitez affecter) votre projet doit aussi être ouvert.

Structure des programmes Visual Basic

Si vous avez déjà développé dans des langages dérivés du Basic, tels que le C, lorsque vous pensez programme, vous imaginez *un* listing dans lequel est contenu l'ensemble du code exécutable du programme. Il n'en va pas de même avec Visual Basic. Les projets VBA sont constitués d'objets distincts, dont la réunion constitue un programme entier.

Les modules

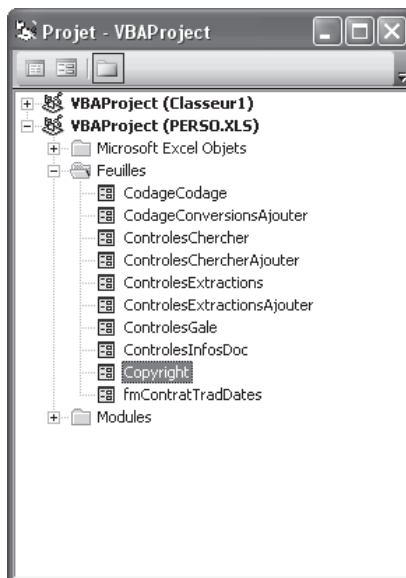
Comme vous l'avez vu en découvrant l'Explorateur de projet au chapitre précédent, on distingue, dans un projet, les modules standard ou modules de code, les modules de classe et les feuilles. Autrement dit, les différents composants du code d'un projet VBA sont structurés et distingués selon leur type. Ces éléments interagissent et s'appellent pour constituer un programme complet.

Le code décrivant l'interface d'un programme et le code affecté aux différents événements qui peuvent toucher cette interface (un clic de souris sur un bouton OK, par exemple) sont stockés dans un fichier UserForm. Pour chaque feuille d'un projet, il existe un objet UserForm accessible dans le dossier Feuilles de l'Explorateur de projet. Vous pouvez voir à la Figure 5.1 que le projet affiché contient onze feuilles.

Le code standard se trouve dans des modules de code, stockés dans le dossier Modules, tandis que le code décrivant les objets développés pour votre projet est stocké dans le dossier Modules de classe.

Figure 5.1

À chaque feuille d'un projet est affecté un fichier dans le dossier Feuilles.



Les procédures

À l'intérieur d'un même module, le code est structuré en *procédures*. Une procédure est une séquence d'instructions s'exécutant en tant qu'entité. Cette décomposition en procédures rend le code plus performant et plus lisible.

Par exemple, lorsqu'un projet VBA ouvre une boîte de dialogue (une feuille), pour chaque événement déclenché par l'utilisateur, l'application vérifie s'il existe une procédure (une unité de code) affectée à cet événement dans le module correspondant. Si tel est le cas, la procédure est exécutée.

C'est l'ensemble des procédures d'un projet et leurs interactions qui forme un programme complet. Par exemple, l'événement clic de souris sur un bouton OK d'une boîte de dialogue peut déclencher une procédure qui récupère et traite l'ensemble des informations contenues dans cette boîte de dialogue (Cette case est-elle cochée ? Quel est le texte saisi dans cette zone de texte ?, etc.). Cette procédure peut ensuite *appeler* (ou *invoquer*) une autre procédure stockée dans le Module de code du projet et lui *passer* les informations ainsi traitées. On parle alors de *procédure appelée* et de *procédure appelante*. La procédure appelée effectuera les tâches pour lesquelles elle a été écrite en exploitant les données fournies par la procédure appelante. Les informations fournies par la procédure appelante à la procédure appelée sont les *arguments passés*.



On qualifie de procédure événementielle ou procédure d'événement une procédure déclenchée par un événement, telle qu'un clic de souris ou la frappe d'une touche clavier, par opposition aux procédures standard d'un module de code, indépendantes de tout événement utilisateur.



Dans les modules UserForm, les procédures sont prédéterminées ; il existe une procédure pour chaque événement pouvant affecter un contrôle. Dans les modules de code, c'est vous qui déterminez les différentes procédures. Vous pouvez ainsi choisir que le code d'un programme soit contenu dans une seule procédure. Il est cependant conseillé de structurer le code d'un programme en procédures distinctes. La lecture et le débogage de vos programmes en seront considérablement améliorés.

Nous vous conseillons de structurer vos programmes sous forme de petites procédures effectuant chacune un traitement spécifique dans le cadre de votre programme. Le programme principal peut alors consister en une procédure qui appelle les autres procédures nécessaires. Examinez le programme suivant :

```
Sub CalculPaieNette()
    NomRepr = InputBox("Entrez le nom du représentant :")
    Call VerifierNomRepresentant(NomRepr)
    Salaire = QuelSalaire(NomRepr)
    ChiffreRepr = InputBox("Entrez le chiffre d'affaires réalisé :")
    Prime = CalculPrime(ChiffreRepr)
    MsgBox "La paie nette sera de " & CalculPaie(SalaireRepr, Prim)
End Sub
```

Cette procédure est composée de six instructions dont quatre font appel à d'autres procédures pour exécuter des tâches spécifiques. La procédure `VerifierNomRepresentant` est appelée afin de vérifier que le nom indiqué est valide. Si le nom n'est pas valide, cette procédure pourra prendre en charge la résolution du problème ou appeler une autre procédure conçue dans ce but. Les procédures `QuelSalaire` et `CalculPrime` sont ensuite appelées afin de renvoyer le salaire et la prime du représentant. Enfin, la procédure `CalculPaie` est appelée pour renvoyer la paie du représentant, qui s'affiche dans une boîte de dialogue.

Les techniques d'appels de procédures sont traitées en détail plus loin dans ce chapitre.

Cette façon de procéder présente plusieurs avantages :

- Les programmes sont plus faciles à lire et à comprendre. Chaque procédure prenant en charge une tâche spécifique du programme, il est plus aisé d'en comprendre le fonctionnement.

- En cas de bogue du programme, il est plus facile d'isoler la procédure coupable et de corriger le problème.
- Le code est ainsi réutilisable. Certaines tâches sont exécutées par différents programmes. Si la tâche a été isolée dans une petite procédure, elle peut être exploitée par différents programmes.

Dans l'exemple précédent, la procédure `VerifierNomRepresentant` est appelée pour contrôler la validité du nom entré par l'utilisateur. Cette procédure peut être appelée par d'autres programmes, sans qu'il soit nécessaire de la réécrire.

Les instructions

Une procédure est composée d'instructions. Chaque instruction exécute une tâche précise, qui peut être évidente ou invisible pour l'utilisateur, destinée à effectuer des traitements propres au projet.

Une instruction Visual Basic est composée de *mots clés* du langage, de constantes et de variables. Ces éléments peuvent être combinés pour former une *expression*. Une expression peut vérifier des données ou effectuer une tâche.

Par exemple, `Mavar = Workbooks.Count` est une instruction. Elle combine la variable `Mavar`, et les mots clés `=`, `Workbooks` et `Count`. Pour affecter à la variable `Mavar` une valeur égale au nombre de classeurs ouverts, on associe cette variable à l'expression `Workbooks.Count` à l'aide de l'opérateur arithmétique `=`. (La propriété `Workbooks` renvoie la collection d'objets `Workbooks` représentant l'ensemble des classeurs ouverts ; la propriété `Count` renvoie le nombre d'objets de la collection spécifiée.)



Un mot clé est un mot ou un symbole reconnu comme élément du langage Visual Basic. Il peut s'agir d'une fonction, d'une propriété, d'une méthode ou encore d'un opérateur arithmétique.



Les constantes sont des éléments nommés affectés à une valeur qui, contrairement à une variable, ne change pas durant l'exécution du programme. Le nom de la constante est utilisé à la place de la valeur qui lui est affectée. Un programme peut exploiter des constantes propres à l'application ou définir ses propres constantes à l'aide de l'instruction `Const` (voir Chapitre 6).

Vous pouvez par exemple définir une constante que vous nommerez `TVA` et à laquelle vous affecterez la valeur 0,186. Chaque fois que vous souhaitez

utiliser cette valeur dans votre programme, il vous suffira d'utiliser le nom de la constante qui lui est affectée. Ainsi, l'expression :

*PrixHorsTaxe * TVA*

sera équivalente à

*PrixHorsTaxe * 0.186*



Les variables sont définies par un nom autre qu'un mot clé du langage et permettent de stocker des informations pouvant être modifiées au cours de l'exécution du programme.

On distingue trois types d'instructions :

- **Les instructions de déclaration.** Ces instructions sont invisibles pour l'utilisateur et servent à nommer une variable, une constante ou une procédure. Le nom attribué à un élément dans l'instruction de déclaration sera ensuite utilisé pour invoquer cet élément tout au long du projet. Une instruction de déclaration peut aussi déterminer le type de l'élément déclaré.

Sub MaProcédure() est un exemple d'instruction de déclaration, nommant la procédure MaProcédure. Dim Mavar As String est aussi une instruction de déclaration. L'instruction Dim sert à nommer la variable Mavar, tandis que As String en spécifie le type (String, c'est-à-dire une chaîne de caractères).

- **Les instructions d'affectation.** Ces instructions affectent une valeur ou une expression à une variable, à une constante ou encore à une propriété – et contiennent donc toujours l'opérateur =. L'exécution de ce type d'instructions peut être visible comme invisible pour l'utilisateur.

MaVar = 5 et MaVar = Workbooks.Count sont des exemples d'instructions d'affectation, attribuant respectivement une valeur et une expression à la variable MaVar.

L'instruction Let est l'instruction d'affectation de Visual Basic. Ainsi l'instruction MaVar = 5 peut aussi être écrite sous la forme Let MaVar = 5. Cependant, cette instruction est facultative et généralement omise.

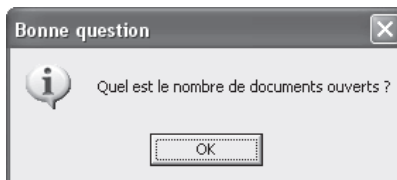
ActiveSheet.Range("C1").Font.Name = "Arial" est une instruction d'affectation visible pour l'utilisateur puisqu'elle applique la police Arial à la cellule C1 de la feuille Excel active – la valeur "Arial" est affectée à la propriété Name de l'objet Font de cette cellule.

- Les instructions exécutables. Ces instructions accomplissent des actions (exécution d'une méthode, d'une fonction). Les instructions de contrôle (traitées au Chapitre 7) sont aussi considérées comme des instructions exécutables.

MsgBox "Quel est le nombre de classeurs ouverts ?", vbOKOnly & vbInformation, "Bonne question" est une instruction exécutable entraînant l'affichage de la boîte de dialogue représentée à la Figure 5.2. L'instruction MsgBox est étudiée au Chapitre 7.

Figure 5.2

Les instructions MsgBox sont des instructions exécutables.



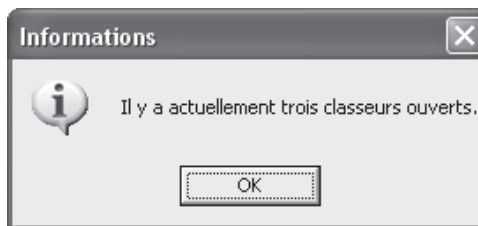
La procédure suivante illustre l'utilisation conjointe des différents types d'instructions :

```
Sub AfficherNbreClasseursOuverts()  
    Dim NbreClasseurs As Byte  
    NbreClasseurs = Workbooks.Count  
    MsgBox "Il y a actuellement" & NbreClasseurs & _  
        "classeurs ouverts.", vbOKOnly + vbInformation, "Informations"  
End Sub
```

Cette procédure affiche une boîte de dialogue indiquant le nombre de classeurs ouverts dans la session Excel active (voir Figure 5.3).

Figure 5.3

Dans une procédure, les instructions exécutables sont les seules instructions visibles pour l'utilisateur.



Les deux premières instructions sont des instructions de déclaration, nommant successivement la procédure AfficherNbreClasseursOuverts et la variable NbreClasseurs. La valeur représentant le nombre de documents ouverts est ensuite affectée à la variable NbreClasseurs dans l'instruction d'affectation de la ligne suivante. Enfin, l'instruction d'exécution MsgBox affiche une boîte de dialogue indiquant à l'utilisateur le nombre de classeurs ouverts dans la session Excel. L'instruction de déclaration End Sub signale la fin de la procédure.

Les différents types de procédures

Une procédure peut exécuter une tâche visible pour l'utilisateur comme effectuer des tâches invisibles, internes au projet, et qui seront exploitées ultérieurement par d'autres procédures.

On distingue différents types de procédures, en fonction du type de tâche qu'exécute la procédure. Les types essentiels de procédures sont :

- les procédures de type Sub ;
- les procédures de type Property ;
- les procédures de type Function.

Procédures *Sub*

Une procédure Sub (ou sous-routine) est une série d'*instructions* exécutant une tâche déterminée au sein du projet, sans renvoyer de valeur. Une procédure Sub est structurée de la façon suivante :

```
Sub NomDeLaProcédure()  
    Instructions  
    ...  
End Sub
```

Les instructions Sub et End Sub déterminent le début et la fin de la procédure. *NomDeLaProcédure* correspond au nom attribué à votre procédure et doit respecter les règles d'affectation de noms de Visual Basic présentées dans la note suivante. Ce nom est utilisé pour invoquer la procédure à partir d'une autre procédure. Dans le cas d'une macro enregistrée ou créée à partir d'Excel, il s'agit du nom attribué à la macro dans la boîte de dialogue Macro ou Enregistrer une macro. *Instructions* correspond aux instructions qu'exécute la procédure.



Les noms de procédures, comme les noms de variables et de constantes, doivent obéir aux règles d'affectation de noms de Visual Basic. Ils doivent systématiquement :

- contenir au maximum 255 caractères ;
- commencer par une lettre ;
 - ne pas comprendre d'espace et ne pas utiliser les caractères @, &, \$, #, ainsi que le point et le point d'exclamation ;
- ne pas être identique à un mot clé du langage, pour éviter tout conflit.

L'instruction de déclaration Sub peut aussi contenir des arguments optionnels, selon la syntaxe suivante :

```
[Private | Public] [Static] Sub NomDeLaProcédure([Arguments])  
    Instructions  
    ...  
End Sub
```

Private ou Public indique si la procédure est *privée* ou *publique*. Une procédure publique peut être invoquée par n'importe quelle procédure du projet, y compris les procédures stockées dans d'autres modules. Une procédure privée ne peut être invoquée qu'à partir d'une procédure stockée dans le même module. On parle de *portée* de la procédure. Lorsque les mots clés Private et Public sont omis dans la déclaration de procédure, la procédure est publique.

L'option Static indique que les variables de la procédure Sub conservent leurs valeurs entre les différents appels de la procédure. Autrement dit, si une procédure Static est invoquée à plusieurs reprises lors de l'exécution d'un programme, à chaque appel, les variables propres à cette procédure ont la valeur qui leur a été affectée lors de l'appel précédent.

Arguments représente les arguments (des valeurs) passés à la procédure Sub par la procédure appelante. La virgule sert de séparateur entre les différentes valeurs transmises.

Dans l'instruction Sub suivante :

```
Sub MaProcédure(arg1, arg2)
```

La procédure MaProcédure est déclarée comme nécessitant les arguments arg1 et arg2. Autrement dit, lorsque la procédure MaProcédure sera appelée, la procédure appelante devra lui transmettre ces arguments ou une erreur sera générée. Les appels de procédures et le passage d'arguments sont étudiés dans la section "Appel et sortie d'une procédure", plus loin dans ce chapitre.

Chacun des arguments de *arguments* répond à la syntaxe suivante :

```
[Optional] [ByVal | ByRef] [ParamArray] NomVariable [As type] [=ValeurParDéfaut]
```



Si vous indiquez des arguments facultatifs à l'aide du mot clé *Optional* dans la déclaration d'une procédure Sub sans spécifier une valeur par défaut pour les arguments concernés, la procédure doit être conçue pour s'exécuter sans faire appel à ces arguments lorsqu'ils ne sont pas transmis. Dans le cas contraire, une erreur sera générée.

Tableau 5.1 : Déclaration d'arguments dans une instruction *Sub*

Élément	Description
Optional	Ce mot clé indique que les arguments transmis sont facultatifs. Aucune erreur ne sera générée si la procédure appelante ne passe pas les arguments précédés de ce mot clé. Lorsque vous déclarez un argument Optional, les arguments qui suivent doivent aussi être déclarés Optional. Si vous souhaitez déclarer des arguments facultatifs et d'autres obligatoires, vous devez d'abord déclarer les arguments obligatoires. Dans l'instruction Sub suivante Sub MaProcédure(arg1, arg2, Optional arg3, Optional arg4), les arguments arg1 et arg2 doivent obligatoirement être passés par la procédure appelante, tandis que les arguments arg3 et arg4 sont facultatifs.
ByVal ou ByRef	Ces mots clés indiquent respectivement que l'argument est passé <i>par valeur</i> ou <i>par référence</i>. Lorsqu'un argument est passé par valeur, c'est la valeur de la variable, et non son adresse, qui est transmise à la procédure appelée. Autrement dit, la valeur de la variable passée est exploitée par la procédure appelée, mais ne peut être modifiée. Lorsqu'un argument est passé par référence, c'est l'adresse de la variable qui est passée. La procédure exploite alors la valeur de la variable transmise, mais peut aussi la modifier. Par défaut (lorsque ces mots clés sont omis), le passage d'un argument se fait par référence. Dans l'exemple suivant : <pre>Sub MaProcédure(ByVal MaVar) MaVar = MaVar * 100 End Sub</pre> La valeur de la variable passée MaVar est multipliée par 100 dans la procédure MaProcédure, mais la valeur réelle de MaVar étant passée par valeur, sa valeur réelle ne sera pas modifiée.
ParamArray	Ce mot clé indique que l'argument est un tableau Optional. L'argument déclaré avec ce mot clé doit être en dernière position dans la liste des arguments et ne peut utiliser conjointement l'un des mots clés précédemment décrits dans ce tableau.
As Type	Spécifie le type de données de l'argument passé à la procédure. Il peut s'agir de données de type Byte, Boolean, Integer, Long, Currency, Single, Double, Date, String, Object, Variant, ou d'un type défini par l'utilisateur. Une erreur est générée si le type de la variable passée par la procédure appelante est incompatible avec le type déclaré. L'instruction de déclaration de MaProcédure suivante indique qu'un argument Arg1 de type String (chaîne de caractères) est requis : <pre>Sub MaProcédure(Arg1 As String)</pre> Les différents types de données ainsi que la création de types de données sont étudiés au Chapitre 6.

<i>Élément</i>	<i>Description</i>
<code>= ValeurParDéfaut</code>	Indique une valeur par défaut pour l'argument. Ce paramètre ne peut être utilisé que conjointement avec le mot clé <code>Optional</code>. Il permet ainsi de déterminer une valeur par défaut qui sera employée si l'argument n'est pas passé par la procédure appelante. Il peut s'agir d'une constante (valeur numérique, chaîne de caractères ou valeur booléenne) ou d'une expression constante. Une expression renvoyant une valeur variable ne peut être utilisée. Dans l'exemple suivant, si l'argument <code>Arg1</code> n'est pas passé par la procédure appelante, sa valeur sera la chaîne "Bonjour". <pre>Sub MaProcédure(Arg1 = "Bonjour")</pre>

Le programme suivant illustre l'utilisation du mot clé `Static` dans une instruction `Sub` :

```
Dim MaVar
Sub ProcédureAppelante()
    MaVar = 2
    Call ProcédureStatic(MaVar)
    Call ProcédureStatic(MaVar)
    Call ProcédureStatic(MaVar)
End Sub

Static Sub ProcédureStatic(MaVar)
    Dim MaVarStatique
    MaVarStatique = MaVarStatique + MaVar
    MsgBox MaVarStatique
End Sub
```

La procédure `ProcédureAppelante` invoque à trois reprises la procédure `ProcédureStatic` à l'aide de l'instruction `Call` en lui passant la variable `MaVar` dont la valeur est 2. À chaque appel, la procédure `ProcédureStatic` incrémente la variable `MaVarStatique` de la valeur de `MaVar` et affiche sa valeur dans une boîte de dialogue à l'aide de la fonction `MsgBox`. La procédure appelante reprend alors la main (et invoque à nouveau la procédure `ProcédureStatic`).

Ce programme affiche donc à trois reprises une boîte de dialogue dont le message est successivement 2, 4, puis 6. En effet, la procédure `ProcédureStatic` étant déclarée `Static`, la variable locale `MaVarStatique` conserve sa valeur entre deux appels. Cette valeur est incrémentée de 2 à chaque exécution de la procédure.



Une variable est dite locale pour une procédure lorsqu'elle est propre à la procédure, par opposition à une variable publique ou passée par la procédure appelante.

Si vous supprimez l'instruction `Static` de la déclaration de procédure `Static Sub ProcédureStatic`, la variable `MaVarStatique` ne conservera plus sa valeur entre les différents appels de procédures, et le programme affichera à trois reprises une boîte de dialogue dont le message sera toujours 2.

Le programme suivant illustre le passage d'un argument par valeur. Les éléments ajoutés par rapport à l'exemple précédent apparaissent en gras :

```
Dim MaVar
Sub ProcédureAppelante()
    MaVar = 2
    Call ProcédureStatic(MaVar)
    Call ProcédureStatic(MaVar)
    Call ProcédureStatic(MaVar)
End Sub

Static Sub ProcédureStatic(ByVal MaVar)
    Dim MaVarStatique
    MaVarStatique = MaVarStatique + MaVar
    MsgBox MaVarStatique
    MaVar = 100
End Sub
```

Ce programme se comporte donc comme nous l'avons détaillé dans l'exemple précédent, mais la procédure `ProcédureStatic` affecte en plus la valeur 100 à `MaVar` avant de redonner la main à la procédure appelante. Malgré cela, le programme continue d'afficher trois boîtes de dialogue dont le message est successivement 2, 4 et 6. En effet, l'instruction de déclaration de `ProcédureStatic` spécifie que l'argument `MaVar` est appelé par valeur (`ByVal`), et non par adresse. La valeur de `MaVar` est donc exploitée, mais sa valeur réelle ne peut être modifiée. `MaVar` reprend donc sa valeur initiale dès que la procédure appelante reprend la main.

Supprimez le mot clé `ByVal` dans l'instruction de déclaration de `ProcédureStatic`. L'argument `MaVar` est maintenant passé par adresse et sa valeur peut donc être modifiée. Le programme ainsi modifié affiche trois boîtes de dialogue dont le message est successivement 2, 102 et 202, reflétant ainsi la modification de la valeur de `MaVar` effective lors des deuxième et troisième exécutions de la procédure `ProcédureStatic`.

Procédures *Function*

Une procédure *Function* (ou fonction) est une série d'*instructions* exécutant une tâche déterminée au sein du projet et renvoyant une valeur, qui sera exploitée par d'autres procédures. Une procédure *Function* est structurée de la façon suivante :

```
Function NomDeLaProcédure(Arguments)  
    Instructions  
    ...  
    NomDeLaProcédure = Expression  
    ...  
End Function
```

Les instructions *Function* et *End Function* déterminent le début et la fin de la procédure. *NomDeLaProcédure* correspond au nom attribué à la procédure et doit respecter les règles d'affectation de noms de Visual Basic. Ce nom est utilisé pour invoquer la procédure à partir d'une autre procédure.

Instructions correspond aux instructions qu'exécute la procédure.

NomDeLaProcédure = *Expression* affecte une valeur à la fonction. *NomDeLaProcédure* est le nom affecté à la procédure dans l'instruction de déclaration *Function*, et *Expression*, une expression affectant une valeur à la fonction. Cette instruction d'affectation peut apparaître à plusieurs reprises et n'importe où dans le code de la fonction. La valeur renvoyée par la fonction est alors la dernière valeur reçue lorsque la fonction prend fin.

L'instruction de déclaration *Function* peut aussi contenir des arguments facultatifs, selon la syntaxe suivante :

```
[Private | Public] [Static] Function NomDeLaProcédure ([Arguments]) [As Type]  
    Instructions  
    ...  
    NomDeLaProcédure = Expression  
End Function
```

As Type permet de préciser le type de valeur renvoyé par la procédure. Il peut s'agir de l'un des types de variables présentés au Chapitre 6. À défaut d'être précisé, la fonction renverra une valeur de type *Variant*. Comme nous l'avons déjà mentionné, il est recommandé de préciser cet argument afin de limiter la mémoire employée par le programme.

Les mots clés *Private* et *Public* indiquent si la procédure est privée ou publique. Le mot clé *Static* indique que les variables locales de la procédure sont statiques, c'est-à-dire conservent leur valeur entre les appels. Pour un rappel de ces concepts, reportez-vous à la section précédente, "Procédures *Sub*".

Arguments représente les arguments passés à la procédure *Function* par la procédure appelante. La virgule sert de séparateur entre les différentes variables transmises.

Dans l'instruction *Function* suivante :

```
Sub MaFonction(arg1, arg2)
```

La procédure *MaFonction* est déclarée comme nécessitant les arguments *arg1* et *arg2*. Autrement dit, lorsque la procédure *MaFonction* sera appelée, la procédure appelante devra lui passer ces arguments ou une erreur sera générée. Les appels de procédures et le passage d'arguments sont étudiés dans la section "Appel et sortie d'une procédure", plus loin dans ce chapitre.

Chacun des arguments de *Arguments* répond à la syntaxe suivante :

```
[Optional] [ByVal | ByRef] [ParamArray] NomVariable [As type] [=ValeurParDéfaut]
```

Cette syntaxe est la même que celle utilisée pour les arguments d'une instruction *Sub*. Pour un rappel de l'utilisation des différents éléments de cette syntaxe, reportez-vous au Tableau 5.1.

La procédure *Function* suivante calcule la surface d'un cercle :

```
Function SurfaceCercle(Rayon As Long) As Long
    Const Pi = 3.14
    SurfaceCercle = Pi * Rayon * Rayon
End Function
```

La première ligne déclare la fonction à l'aide de l'instruction *Function* en indiquant que l'argument *Rayon* est requis. Cet argument devra donc être précisé dans l'instruction appelante. La fonction ainsi que l'argument attendu sont de type *Long*. Une constante *Pi* est ensuite définie. À la troisième ligne, la fonction *SurfaceCercle* est affectée à une expression représentant la surface du cercle d'un rayon de *Rayon* centimètres. Enfin, l'instruction *End Function* signale la fin de la procédure.

Les procédures *Function* peuvent très simplement être appelées à partir d'autres procédures. Lorsqu'une procédure *Function* est définie, vous pouvez l'utiliser comme n'importe quelle fonction intégrée de Visual Basic, c'est-à-dire en faisant apparaître son nom suivi de la liste des arguments requis par la fonction entre parenthèses, dans une expression.

Considérez le programme suivant :

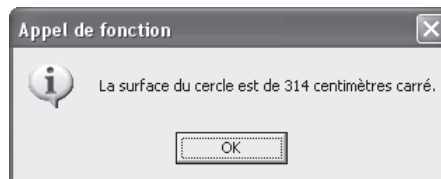
```
Sub MaProcédure()
    Dim Rayon
    Rayon = 10
```

```
MsgBox "La surface du cercle est de" & SurfaceCercle(Rayon) & "centimètres  
➡ carré.", _  
vbOKOnly & vbInformation, "Appel de fonction"  
End Sub  
  
Function SurfaceCercle(Rayon)  
Const Pi = 3.14  
SurfaceCercle = Pi * Rayon * Rayon  
End Function
```

La sous-routine MaProcédure déclare la variable Rayon et lui affecte la valeur 10. Elle affiche ensuite une boîte de dialogue dont une partie du message (en gras, dans le listing) fait appel à la fonction SurfaceCercle en lui passant l'argument Rayon précédemment défini. La fonction SurfaceCercle calcule donc la surface du cercle et rend la main à la procédure appelante qui affiche la boîte de dialogue présentée à la Figure 5.4.

Figure 5.4

L'instruction MsgBox fait appel à la fonction SurfaceCercle pour afficher la surface du cercle.



Procédures *Property*

Une procédure Property (ou procédure de propriété) est une série d'*instructions* exécutant une tâche déterminée au sein du projet et manipulant des données de type Propriétés. Il existe trois types de procédures Property :

- **Property Get.** Elles renvoient la valeur d'une propriété qui sera ensuite exploitée par d'autres procédures.
- **Property Let.** Ces procédures définissent la valeur d'une propriété.
- **Property Set.** Elles établissent une référence entre un objet et une propriété.



Une propriété est un attribut nommé d'un objet, définissant ses caractéristiques ou son état. Par exemple, la propriété Address d'un objet Range renvoie l'adresse de cet objet (A1, par exemple). La propriété ColorIndex d'un objet Font renvoie la couleur de cet objet (une police de caractères).

Procédures *Property Get*

Une procédure *Property Get* est structurée de la façon suivante :

```
Property Get NomDeLaProcédure ()  
    Instructions  
    ...  
    NomDeLaProcédure = Expression  
    ...  
End Property
```

Les instructions *Property Get* et *End Property* déterminent le début et la fin de la procédure. *NomDeLaProcédure* correspond au nom attribué à la procédure et doit respecter les règles d'affectation de noms de Visual Basic. Ce nom est utilisé pour invoquer la procédure à partir d'une autre procédure.

Instructions correspond aux instructions qu'exécute la procédure.

NomDeLaProcédure = *Expression* affecte une valeur à la procédure de propriété. *NomDeLa-Procédure* est le nom affecté à la procédure dans l'instruction de déclaration *Property Get*, et *Expression*, une expression affectant une valeur à la procédure. Cette instruction d'affectation peut apparaître à plusieurs reprises et n'importe où dans le code de la procédure.

L'écriture d'une procédure *Property Get* se justifie lorsque la valeur de la propriété ne peut être renvoyée en une seule instruction Visual Basic – par exemple, lorsque l'on souhaite renvoyer sous forme de chaîne de caractères une propriété affectée à une constante.

La procédure de propriété suivante renvoie une chaîne de caractères représentant un commentaire fonction de la valeur de la cellule qui lui est passée. Cette information est renvoyée par la propriété *Value* de l'objet *Cellule*. En fonction de la valeur renvoyée, une chaîne de caractères est affectée à la procédure.

```
1: Property Get RenvoyerCommentaire(Cellule As Range) As String  
2:     Select Case Cellule.Value  
3:         Case Is < 10000  
4:             RenvoyerCommentaire = "Très mauvais"  
5:         Case 10000 To 20000  
6:             RenvoyerCommentaire = "Mauvais"  
7:         Case 20001 To 30000  
8:             RenvoyerCommentaire = "Correct"  
9:         Case 30001 To 40000  
10:            RenvoyerCommentaire = "Bon"  
11:         Case Is > 40000  
12:            RenvoyerCommentaire = "Très bon"  
13:     End Select  
14: End Property
```



Ce listing est numéroté de façon à simplifier la présentation des différentes instructions de la procédure RenvoyerCommentaire. La présence de cette numérotation dans la procédure réelle générerait évidemment une erreur à l'exécution.

À la ligne 1, l'instruction `Property Get` déclare la procédure `RenvoyerCommentaire`, qui doit recevoir l'argument `Cellule` de type `Range` (`Cellule As Range`), et qui renvoie une valeur de type `String` (`As String`). Une instruction de contrôle `Select Case` est utilisée pour tester la valeur renvoyée par l'expression `Cellule.Value` des lignes 2 à 13. Pour chacune des plages de valeurs testées, une chaîne de caractères est affectée à la procédure de propriété (`RenvoyerCommentaire = "commentaire"`).

Les procédures `Property Get` peuvent très simplement être appelées à partir d'autres procédures. Lorsqu'une procédure `Property Get` est définie, vous pouvez l'utiliser dans n'importe quelle expression exploitant une valeur de propriété, c'est-à-dire en faisant apparaître son nom suivi de la liste des éventuels arguments entre parenthèses.

Considérez les procédures suivantes :

```
Sub DefinirCommentaire()
    Dim LaCellule As Range
    For Each LaCellule In Selection
        LaCellule.AddComment (RenvoyerCommentaire(LaCellule))
    Next LaCellule
End Sub

Property Get RenvoyerCommentaire(Cellule) As String
    Select Case Cellule.Value
        Case Is < 10000
            RenvoyerCommentaire = "Très mauvais"
        Case 10000 To 20000
            RenvoyerCommentaire = "Mauvais"
        Case 20001 To 30000
            RenvoyerCommentaire = "Correct"
        Case 30001 To 40000
            RenvoyerCommentaire = "Bon"
        Case Is > 40000
            RenvoyerCommentaire = "Très bon"
    End Select
End Property
```

La sous-routine `DefinirCommentaire` appelle la procédure de propriété `RenvoyerCommentaire`. Une variable objet de type `Range` y est créée, et une structure `For Each...Next` est utilisée pour effectuer un traitement sur chaque cellule de la sélection en cours dans la feuille de calcul active. N'essayez pas pour l'instant de comprendre ces instructions.

Les variables et les structures de contrôle – telles que les structures For Each Next et Select Case – sont présentées en détail dans les deux chapitres suivants.

La méthode AddComment est appliquée à chaque cellule afin d'insérer un commentaire. Cette méthode s'utilise selon la syntaxe suivante :

Expression.AddComment(*Texte*)

où *Expression* est une expression renvoyant un objet Range (une cellule ou une plage de cellules) et *Texte*, le texte du commentaire.

Plutôt qu'une chaîne de caractères, l'argument *Texte* reçoit ici pour valeur l'expression RenvoyerCommentaire(LaCellule). Cette expression appelle la procédure Property Get du même nom, en lui passant l'argument LaCellule. Cette procédure s'exécute et renvoie une chaîne de caractères. Celle-ci est passée à la procédure appelante qui l'affecte comme commentaire de la cellule (voir Figure 5.5).

Figure 5.5

Chacune des cellules comprises dans la sélection au moment de l'exécution de la macro s'est vue affecter un commentaire.

	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00	54 897,00	
6	Mai	45 258,00	32 568,00	36 550,00		
7	Juin	38 652,00	97 632,00	56 320,00		
8	Juillet	32 510,00	45 863,00	45 520,00		
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	

Procédures Property Let

Une procédure Property Let est structurée de la façon suivante :

```
Property Let NomDeLaProcédure (VarStockage)
    Instructions
End Property
```

Les instructions Property Let et End Property déterminent le début et la fin de la procédure. *NomDeLaProcédure* correspond au nom attribué à la procédure et doit respecter les règles d'affectation de noms de Visual Basic. Ce nom est utilisé pour invoquer la procédure à partir d'une autre procédure.

Instructions correspond aux instructions qu'exécute la procédure.

VarStockage est la variable recevant la valeur passée par la procédure appelante et assimilée à la valeur affectée à la propriété. Comme vous le verrez dans l'exemple suivant, cette valeur peut ensuite être traitée par la procédure afin de déterminer la valeur devant être affectée à la propriété.

La procédure *Property Let* suivante est en quelque sorte la procédure miroir de la procédure *Property Get*, décrite plus haut. Cette procédure affecte une couleur de remplissage à chacune des cellules de la sélection en cours, en fonction du commentaire affecté à la cellule. La chaîne représentant le commentaire détermine la valeur de la variable numérique qui sera affectée à la propriété *ColorIndex* de l'objet *Range* (la cellule).

```
1: Property Let CouleurDeRemplissage(LaCellule As Range)
2:   Dim IndexCouleur As Integer
3:   Select Case LaCellule.Comment.Text
4:     Case "Très mauvais"
5:       IndexCouleur = 3 'Index de la couleur Rouge
6:     Case "Mauvais"
7:       IndexCouleur = 6 'Index de la couleur Jaune
8:     Case "Correct"
9:       IndexCouleur = 5 'Index de la couleur Bleu
10:    Case Else
11:      IndexCouleur = xlColorIndexNone
12:    End Select
13:    LaCellule.Interior.ColorIndex = IndexCouleur
14: End Property
```

À la ligne 1, l'instruction *Property Let* déclare la procédure *CouleurDeRemplissage*, spécifiant que l'argument *LaCellule* de type *Range* doit être passé par la procédure appelante. La variable *IndexCouleur* est déclarée à l'aide de l'instruction *Dim* à la ligne 2. Une instruction de contrôle *Select Case* est ensuite utilisée pour tester la valeur de l'expression *LaCellule.Comment.Text*, qui renvoie le texte de commentaire de la cellule (lignes 3 à 12). Si la valeur renvoyée est égale à *Très mauvais*, *Mauvais* ou *Correct*, une valeur représentant un index de couleur est affectée à la variable *IndexCouleur*. Si le commentaire de la cellule ne correspond à aucune de ces valeurs, *IndexCouleur* se voit affecter la constante *xlColorIndexNone*. Enfin, l'instruction de la ligne 13 affecte la valeur de *IndexCouleur* à la propriété *ColorIndex* de l'objet *Interior* de la cellule, c'est-à-dire applique une couleur de remplissage à la cellule.

Pour appeler une procédure *Property Let*, utilisez une instruction d'affectation, assimilable à l'affectation d'une propriété (*Propriété = Valeur*). Le nom de la procédure placé à gauche de l'instruction d'affectation sera assimilé à une expression renvoyant une propriété. L'argument requis par la procédure *Property Let* placée à droite de l'expression sera assimilé à la valeur affectée à la propriété.

Considérez l'exemple suivant :

```
Sub DefinirRemplissage()  
    Dim LaCellule As Range  
    For Each LaCellule In Selection  
        CouleurDeRemplissage = LaCellule  
    Next LaCellule  
End Sub  
  
Property Let CouleurDeRemplissage(LaCellule As Range)  
    Dim IndexCouleur As Integer  
    Select Case LaCellule.Comment.Text  
        Case "Très mauvais"  
            IndexCouleur = 3 'Index de la couleur Rouge  
        Case "Mauvais"  
            IndexCouleur = 6 'Index de la couleur Jaune  
        Case "Correct"  
            IndexCouleur = 5 'Index de la couleur Bleu  
        Case Else  
            IndexCouleur = xlColorIndexNone  
    End Select  
    LaCellule.Interior.ColorIndex = IndexCouleur  
End Property
```

Dans la sous-routine `DefinirRemplissage`, l'instruction d'affectation `CouleurDeRemplissage = LaCellule` est assimilée à une affectation de propriété. `CouleurDeRemplissage` est assimilé à une expression renvoyant une propriété et `LaCellule`, à la valeur affectée à cette propriété. La procédure `CouleurDeRemplissage` est donc appelée par cette instruction. Elle s'exécute et interroge la valeur de la propriété `Text` de l'objet `Comment` de la variable `LaCellule`. La variable `IndexCouleur` reçoit une valeur, fonction du résultat retourné. Cette valeur est ensuite affectée à la propriété `ColorIndex` de l'objet `Interior` de la cellule. La procédure appelante reprend ensuite la main et procède de la même façon pour la cellule suivante de la sélection.

Syntaxe avancée

Au même titre que les instructions `Sub` et `Function`, les instructions de déclaration `Property Get` et `Property Let` peuvent être précédées des mots clés `Private` ou `Public` et/ou `Static`, afin de spécifier si la procédure est privée ou publique, et si ses variables locales sont statiques. Pour un rappel de ces concepts, reportez-vous à la section "Procédures Sub" de ce chapitre.

Les instructions de déclaration `Property Get` et `Property Let` peuvent aussi spécifier des arguments dans les parenthèses suivant le nom de la procédure. Chacun des arguments répond à la syntaxe suivante :

[Optional] [ByVal | ByRef] [ParamArray] *NomVariable* [As *type*] [=ValeurParDéfaut]

Cette syntaxe est la même que celle utilisée pour les arguments des instructions de déclaration `Sub` et `Function`. Pour un rappel de l'utilisation des différents éléments de cette syntaxe, reportez-vous à la section "Procédures Sub", plus haut dans ce chapitre.

Enfin, une instruction `Property Get` peut se terminer par l'argument `As Type` précisant le type de valeur renvoyé par la procédure. Il peut s'agir de l'un des types de variables présentés au Chapitre 6.

Des projets bien structurés

Contrairement aux feuilles qui sont automatiquement stockées dans le dossier `UserForm` de votre projet, il vous incombe de déterminer l'organisation des modules de code et des modules de classe. Vous pouvez ainsi structurer les procédures constituant votre projet, de façon à y accéder facilement.

Les modules sont en quelque sorte les dossiers dans lesquels vous rangez les papiers qui constituent vos applications VBA. Une bonne organisation vous assurera de retrouver rapidement les documents dont vous avez besoin.

Organisez les procédures d'un projet par modules cohérents, en réunissant les procédures qui ont une fonction commune au sein d'un même module. Veillez aussi à structurer le code en procédures distinctes. Si une application VBA peut être contenue dans une seule procédure, la division des tâches complexes en plusieurs procédures distinctes qui s'appelleront est fortement recommandée. Le code ainsi segmenté sera plus facile à gérer, et le débogage considérablement simplifié.

Ajouter un module

Vous serez probablement amené à développer des applications VBA distinctes au sein d'un même projet. Il est alors important de placer les procédures constituant une application au sein de mêmes modules. Si vous stockez toutes les procédures d'un projet au sein d'un même module, sans aucune distinction, vous risquez d'être rapidement dépassé par un nombre important de procédures dont vous serez incapable de définir les rapports.

Pour créer un module standard ou un module de classe, procédez comme suit :

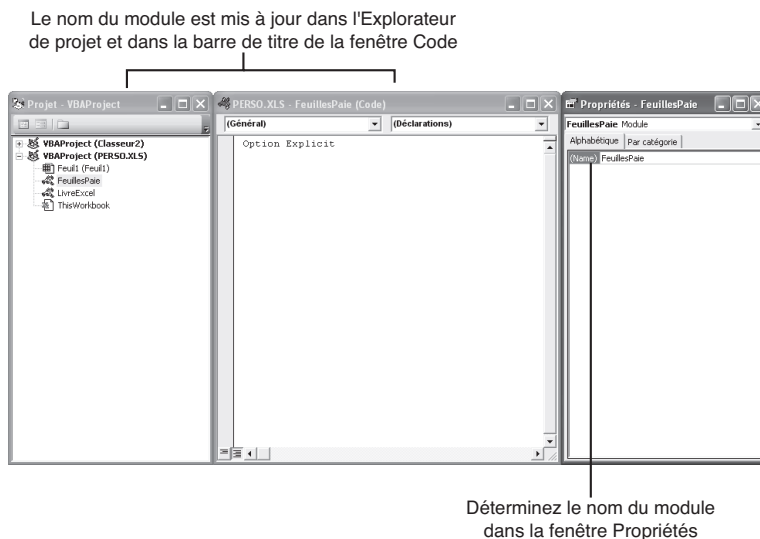
1. Lancez Visual Basic Editor à partir d'Excel (Alt+F11) – le document hôte du projet doit être ouvert.
2. Dans Visual Basic Editor, affichez l'Explorateur de projet (Ctrl+R). Si plusieurs projets sont accessibles dans l'Explorateur de projet, cliquez sur n'importe quel élément du projet auquel vous souhaitez ajouter un module, afin de l'activer.
3. Pour ajouter un module au projet actif, choisissez l'une des trois méthodes suivantes :
 - Cliquez du bouton droit et, dans le menu contextuel qui s'affiche, sélectionnez Insertion. Dans le sous-menu, sélectionnez Module ou Module de classe.
 - Ouvrez le menu Insertion et choisissez la commande Module ou Module de classe.
 - Cliquez sur la flèche du bouton Ajouter... de la barre d'outils Standard. Dans le menu qui s'affiche, sélectionnez Module ou Module de classe.



La fenêtre Code du module inséré s'ouvre. Le module inséré est automatiquement nommé : Module1 (Module2, s'il existe déjà un module nommé Module1...) s'il s'agit d'un module standard, et Class1 (Class2, s'il existe déjà un module nommé Class1...) s'il s'agit d'un module de classe.

4. Pour attribuer un nom au nouveau module, ouvrez sa fenêtre Propriétés (F4). Indiquez un nom représentatif. Le nom du module apparaissant dans la barre de titre de la fenêtre Code et dans l'Explorateur de projet est automatiquement mis à jour (voir Figure 5.6).

Figure 5.6
Choisissez des noms représentatifs pour vos modules.



Supprimer un module

Pour supprimer un module ou une feuille d'un projet, procédez comme suit :

1. Sélectionnez un module ou une feuille dans l'Explorateur de projet.
2. Cliquez du bouton droit et, dans le menu contextuel qui s'affiche, sélectionnez Supprimer *Module* (où *Module* est le nom du module sélectionné).

Visual Basic Editor affiche une boîte de dialogue vous proposant d'exporter le module avant la suppression (voir Figure 5.7).

Figure 5.7

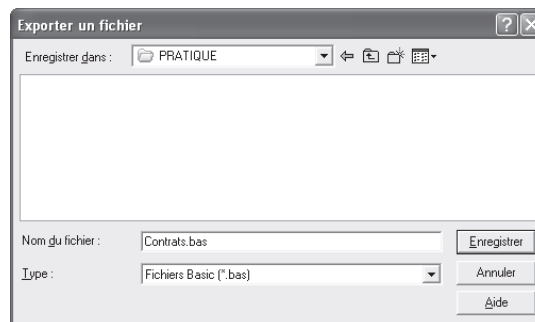
Pour sauvegarder les informations contenues dans un module avant sa suppression, exportez-le.



3. Si vous souhaitez supprimer définitivement le module, choisissez Non. Le module est supprimé.
4. Pour effectuer une sauvegarde du module afin de pouvoir le récupérer en cas de nécessité, choisissez Oui.
5. La boîte de dialogue Exporter un fichier s'affiche (voir Figure 5.8). Le type du fichier exporté varie selon l'élément supprimé :
 - Les modules standard ou modules de code sont exportés sous la forme de fichiers Basic portant l'extension .bas.
 - Les modules de classe sont exportés sous la forme de fichiers Classe portant l'extension .cls.
 - Les feuilles sont exportées sous la forme de fichiers Feuille portant l'extension .frm.
6. Indiquez le répertoire et le nom d'enregistrement, puis cliquez sur le bouton Enregistrer.

Figure 5.8

Indiquez le nom du fichier exporté et son dossier d'enregistrement.





Pour exporter un module sans le supprimer, sélectionnez la commande Exporter un fichier du menu Fichier. Vous pourrez ensuite l'importer dans n'importe quel projet en choisissant Fichier > Importer un fichier.

Créer une procédure

Pour créer une procédure, activez la fenêtre Code du module dans lequel sera stockée la procédure. Si vous souhaitez créer un nouveau module, reportez-vous à la section "Des projets bien structurés", en fin de chapitre.

Vous pouvez ensuite écrire directement l'instruction de déclaration de la procédure ou utiliser la boîte de dialogue Ajouter une procédure.

Écrire l'instruction de déclaration

1. Dans la fenêtre Code du module voulu, placez le point d'insertion à l'endroit où vous souhaitez insérer une procédure.



Les procédures ne peuvent être imbriquées. Veillez donc à ce que le point d'insertion se trouve à l'extérieur de toute procédure.

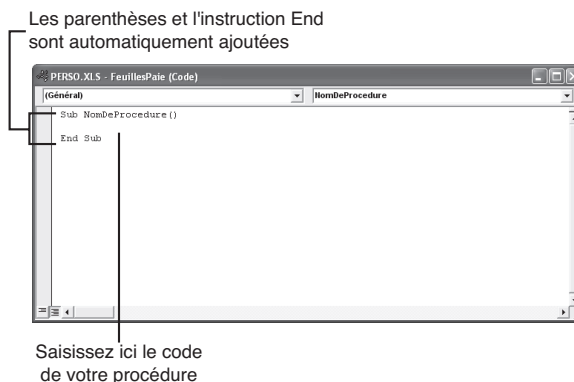
2. Saisissez les éventuels mots clés précisant la portée de la procédure (Public ou Private) et le comportement des variables locales (Static). (Étape facultative.)
3. Tapez l'instruction de déclaration correspondant au type de procédure que vous souhaitez créer, soit Sub, Function, Property Get, Property Let ou Property Set.
4. Saisissez le nom de la procédure.
5. Indiquez ensuite les éventuels arguments de la procédure entre parenthèses. (Étape facultative.)
6. Tapez sur la touche Entrée afin de placer un retour chariot.

Visual Basic Editor ajoute automatiquement l'instruction End correspondant au type de procédure ajouté (End Sub, End Function ou End Property). Si vous n'avez indiqué aucun argument pour la procédure, une parenthèse ouvrante, immédiatement suivie d'une parenthèse fermante, est automatiquement ajoutée derrière le nom de la procédure (voir Figure 5.9).

7. Saisissez le code de votre procédure entre ces instructions d'encadrement.

Figure 5.9

Saisissez l'instruction de déclaration de la procédure, et Visual Basic Editor insérera l'instruction End correspondante.

**Info**

*Vous pouvez aussi préciser le comportement et la disponibilité de la procédure en ajoutant, par exemple, le mot clé **Static** devant l'instruction de déclaration de façon à définir les variables locales comme statiques. Pour plus de précisions, reportez-vous au Chapitre 6.*

La boîte de dialogue Ajouter une procédure

1. Activez la fenêtre Code du module dans lequel vous souhaitez insérer une procédure.
2. Cliquez sur la flèche située à droite du bouton Ajouter... de la barre d'outils Standard et, dans le menu qui s'affiche, sélectionnez la commande Procédure.



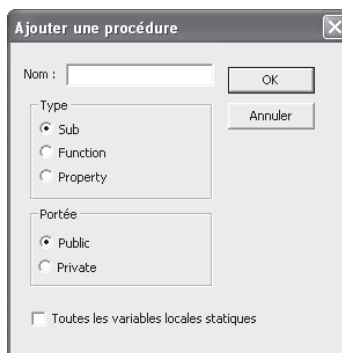
La boîte de dialogue Ajouter une procédure présentée à la Figure 5.10 s'affiche.

Info

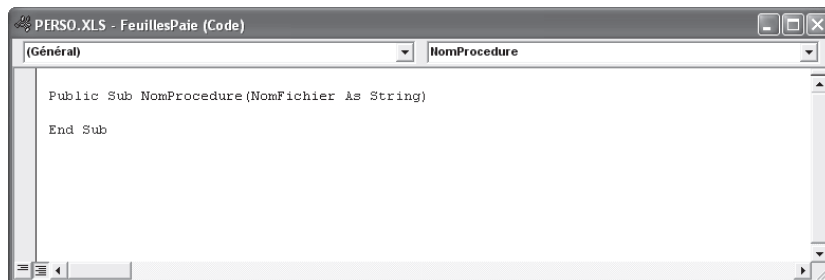
La commande Procédure n'est pas disponible si la fenêtre active n'est pas une fenêtre Code.

Figure 5.10

La boîte de dialogue Ajouter une procédure.



3. Entrez les informations décrivant la procédure que vous souhaitez insérer dans la boîte de dialogue Ajouter une procédure :
 - Dans la zone Nom, saisissez le nom qui sera affecté à la procédure.
 - Dans la zone Type, cliquez sur le bouton radio correspondant au type de procédure que vous souhaitez ajouter.
 - Dans la zone Portée, spécifiez la portée de la procédure. Le mot clé correspondant sera ajouté devant l’instruction de déclaration.
 - Cochez éventuellement la case Toutes les variables locales statiques. Cela ajoute le mot clé Static devant l’instruction de déclaration de la procédure.
4. Cliquez sur le bouton OK. La boîte de dialogue se ferme, et les instructions d’encadrement de la procédure sont insérées dans la partie inférieure de la fenêtre Code.
5. Saisissez les éventuels arguments de la procédure entre les parenthèses situées derrière le nom de la procédure.
6. Saisissez le code de votre procédure entre l’instruction de déclaration de la procédure et l’instruction End correspondante.

**Figure 5.11**

Les instructions d’encadrement de la procédure définie dans la boîte de dialogue Ajouter une procédure sont automatiquement insérées dans la partie inférieure de la fenêtre Code.



Lorsque vous sélectionnez le type Property, les instructions d’encadrement d’une procédure Property Get et d’une procédure Property Let sont insérées.

La notion de portée

La portée d’une procédure est essentielle puisqu’elle en détermine l’accessibilité pour les autres procédures du projet. Les procédures peuvent être *publiques* ou *privées*. Une procédure publique est visible pour n’importe quelle procédure du projet, quel que soit

son module de stockage. Une procédure privée est invisible pour les procédures autres que celles de son module.

Ainsi, une procédure devant être appelée par une procédure appelante située dans un autre module doit être *publique*. Si, au contraire, vous souhaitez interdire l'appel d'une procédure par une procédure appelante située dans un autre module, vous devez la déclarer *privée*.

La portée d'une procédure est déterminée dans son instruction de déclaration. Le mot clé `Public` indique une procédure de portée publique, tandis que le mot clé `Private` indique une procédure de portée privée.

Par défaut, les procédures des modules de code et des modules de classe sont publiques. Il n'est donc pas utile de placer le mot clé `Public` devant une instruction de déclaration de procédure pour la rendre publique. Par contre, si vous souhaitez déclarer une procédure privée, vous devez placer le mot clé `Private` devant l'instruction de déclaration de cette procédure.

Les procédures événementielles sont, par définition, privées. Elles ne peuvent en effet pas être appelées par une autre procédure, puisque seul l'événement spécifié peut les déclencher. Comme vous le verrez au Chapitre 14, lorsque vous créez des procédures événementielles, le mot clé `Private` est automatiquement inséré.

Les deux instructions suivantes déclarent une procédure `Sub` de portée publique :

```
Sub MaProcédurePublique()  
    Instructions  
End Sub  
  
Public MaProcédurePublique()  
    Instructions  
End Sub
```

L'instruction suivante déclare une procédure `Sub` de portée privée :

```
Private MaProcédurePrivée()  
    Instructions  
End Sub
```

Écriture et mise en forme du code

Les instructions constituant une procédure sont autonomes et sont généralement écrites sur une ligne distincte. Pour écrire une instruction, placez le curseur sur une ligne vierge et saisissez le texte au clavier. L'instruction écrite, tapez sur la touche `Entrée` et écrivez l'instruction suivante.

Caractère de continuité de ligne

Une instruction peut cependant être écrite sur plusieurs lignes de code en plaçant le caractère de continuité de ligne – le trait de soulignement (_) – afin d’indiquer que l’instruction se poursuit sur la ligne suivante. Le caractère de continuité de ligne facilite dans certains cas la lecture du code, en permettant de lire une instruction longue sans avoir à se déplacer dans la fenêtre Code à l’aide de la barre de défilement horizontale.

Ainsi, la procédure suivante est composée des instructions de déclaration Sub et End Sub et d’une seule et même instruction exécutable MsgBox répartie sur plusieurs lignes à l’aide du caractère de continuité de ligne :

```
Sub MaProcédure()  
    MsgBox ("Cette instruction affiche une boîte de dialogue" _  
        & "dans laquelle est affiché ce très long message, que nous avons" _  
        & "réparti sur quatre lignes à l'aide du caractère de continuité" _  
        & "de ligne")  
End Sub
```

Notez que le message affiché par la fonction MsgBox a été séparé en plusieurs chaînes de caractères concaténées à l’aide de l’opérateur &. Le caractère de continuité de ligne ne peut en effet être utilisé qu’entre des éléments distincts d’une instruction. Autrement dit, une même chaîne de caractères, une expression, un mot clé ou encore une constante ne peuvent être écrits sur plusieurs lignes.

Les commentaires

Un *commentaire* est une indication destinée à faciliter la lecture du code en décrivant les tâches qu’exécute une instruction, la date de création d’une procédure, etc.

L’insertion de commentaires dans le texte d’une procédure peut se révéler utile, particulièrement si elle comprend un grand nombre d’instructions dont vous souhaitez reconnaître rapidement les fonctions respectives.

Si vous souhaitez insérer des commentaires dans le texte d’une procédure, il faut bien évidemment que ceux-ci soient reconnus en tant que tels, et non en tant qu’instructions, ce qui générerait une erreur lors de l’exécution de la procédure.

Visual Basic considère le texte en tant que commentaire et l’ignore lors de l’exécution d’une procédure s’il est précédé :

- d’une apostrophe (') ;
- du mot clé REM.

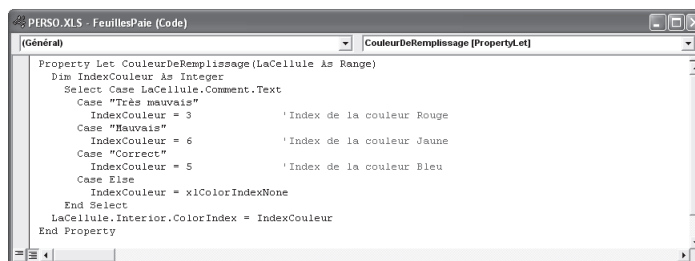
Par défaut, les commentaires apparaissent en vert dans Visual Basic Editor. Pour modifier la couleur d'affichage des commentaires, reportez-vous à la section "Un code tout en couleur", plus loin dans ce chapitre.

Utiliser l'apostrophe

L'utilisation de l'apostrophe pour marquer les commentaires permet de les placer à n'importe quel endroit du texte. Vous pouvez ainsi insérer un commentaire sur la même ligne que l'instruction concernée en plaçant autant d'espaces que vous le souhaitez entre celle-ci et l'apostrophe. L'apostrophe se révélera particulièrement intéressante et efficace pour les instructions dont la syntaxe est courte, puisqu'elle permettra d'aligner les différents commentaires. À la Figure 5.12, l'apostrophe a été utilisée comme marqueur de commentaires dans la macro *GrasItalique* du Chapitre 2.

Figure 5.12

L'apostrophe permet d'aligner les commentaires.

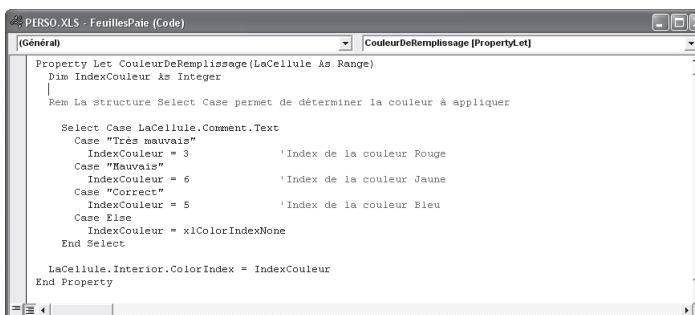


Utiliser REM

La syntaxe REM joue le même rôle que l'apostrophe. Cependant, contrairement à celle-ci, le marqueur REM ne peut être accolé à l'instruction qu'il commente mais doit toujours être placé en début de ligne. Il sera donc utilisé de préférence pour commenter des blocs d'instructions plutôt qu'une seule instruction. La Figure 5.13 illustre l'utilisation de REM.

Figure 5.13

L'instruction REM ne peut être placée qu'en début de ligne.





Vous pouvez utiliser des caractères facilement discernables pour faire ressortir les commentaires. Vous pouvez, par exemple, présenter vos commentaires de la façon suivante :

```
' *****
```

```
'****Ceci est un commentaire****
```

```
' *****
```

Commenter un bloc d'instructions



Lorsque vous testerez le comportement des applications VBA, il se révélera parfois nécessaire de placer en commentaire un bloc d'instructions que vous ne souhaitez pas voir s'exécuter lors de cette phase. Le bouton Commenter bloc de la barre d'outils Édition permet de commenter toutes les instructions sélectionnées dans la fenêtre Code active, en plaçant une apostrophe devant chaque ligne de la sélection.



Pour réactiver un bloc d'instructions commentées, sélectionnez les instructions en question et cliquez sur le bouton Ne pas commenter de la barre d'outils Édition. Toutes les apostrophes de début de ligne du bloc d'instructions sont supprimées.



Lorsque vous enregistrez une macro dans l'application hôte, des lignes de commentaires indiquant la date d'enregistrement de la macro sont placées derrière l'instruction de déclaration de la procédure.

Mise en forme du code

La mise en forme du code consiste à appliquer des retraits de ligne variables aux instructions, et à effectuer d'éventuels sauts de ligne entre les blocs d'instruction afin d'améliorer la lisibilité et de faciliter l'interprétation du code. On applique en général un même retrait aux instructions de même niveau, c'est-à-dire s'exécutant à un même niveau dans la procédure ou appartenant à une même structure. Lorsque des instructions sont imbriquées dans d'autres instructions, on leur applique un retrait supplémentaire par rapport à ces dernières. La hiérarchie qui régit les instructions d'une procédure peut ainsi être visualisée, ce qui en simplifie la lecture (voir Figure 5.14).

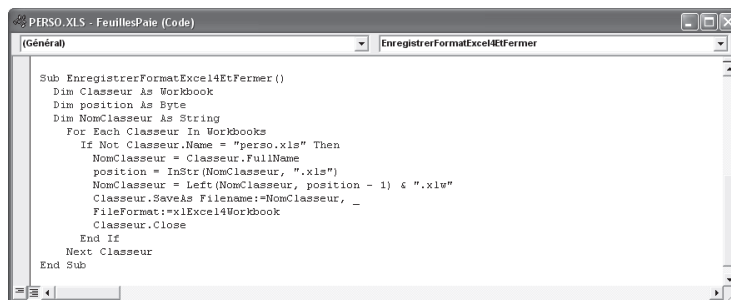


Des instructions sont dites imbriquées lorsque leur exécution s'effectue à l'intérieur d'un autre bloc d'instructions. Vous pouvez, par exemple, créer une instruction conditionnelle qui vérifie qu'une plage de cellules d'une feuille de classeur contient des valeurs numériques, et y imbriquer une autre structure

conditionnelle qui effectuera des tâches déterminées. Les structures de contrôle et leur imbrication sont étudiées au Chapitre 7.

Figure 5.14

Utilisez les retraits de ligne pour améliorer la lisibilité de votre code.



Pour appliquer des retraits de ligne, vous pouvez placer indifféremment des espaces ou des tabulations en début de ligne. L'éditeur peut aussi être paramétré de façon que, lorsqu'un retrait de ligne est appliqué, le même retrait soit appliqué à la ligne suivante :

1. Choisissez la commande Options du menu Outils et sélectionner l'onglet Éditeur de la boîte de dialogue Options.
2. Cochez la case Retrait automatique, puis définissez une valeur de retrait comprise entre 1 et 32 dans la zone Retrait de la tabulation. Cette valeur correspond au nombre d'espaces qui seront appliqués lors de la frappe de la touche Tabulation.
3. Cliquez sur OK pour valider les paramètres définis.

Pour augmenter ou diminuer simultanément le retrait de ligne de plusieurs instructions, sélectionnez celles-ci, puis cliquez sur le bouton Retrait ou sur le bouton Retrait négatif de la barre d'outils Édition. La valeur du retrait appliqué correspond à la valeur définie pour le retrait de tabulation.

Un code tout en couleur

Toujours dans l'optique de faciliter l'interprétation du code, les éléments constitutifs d'une procédure sont affichés dans différentes couleurs, chaque couleur identifiant une catégorie spécifique du langage. Par exemple, les commentaires sont affichés par défaut en vert, tandis que les mots clés du langage apparaissent en bleu et que les erreurs de syntaxe sont affichées en rouge (voir Figure 5.15).

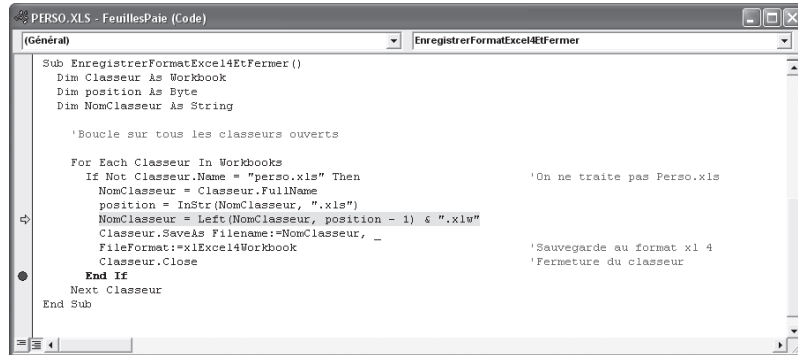


Figure 5.15

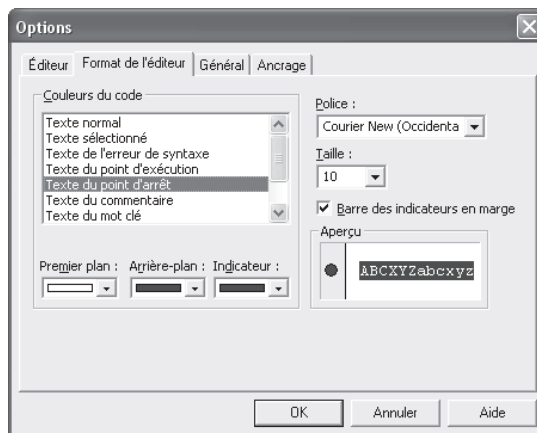
Les couleurs déterminent les catégories de texte et mettent en valeur les erreurs de syntaxe.

Pour définir vos propres paramètres d’affichage du texte dans la fenêtre Code, procédez comme suit :

1. Choisissez la commande Options du menu Outils, puis activez l’onglet Format de l’éditeur de la boîte de dialogue qui s’affiche (voir Figure 5.16).
2. Dans la zone Couleur de code, sélectionnez la catégorie dont vous souhaitez modifier les paramètres d’affichage. Déterminez ensuite les options d’affichage de votre choix dans les zone de liste déroulantes suivantes :
 - **Premier plan.** Définit la couleur de premier plan. Lorsque le texte n’est pas sélectionné, il s’agit de la couleur des caractères.
 - **Arrière-plan.** Détermine la couleur d’arrière-plan.
 - **Indicateurs.** Définit la couleur des indicateurs apparaissant en marge, pour les catégories du langage affichant ce type d’indicateur.
3. Modifiez éventuellement la police d’affichage du code, en sélectionnant la police de caractères de votre choix dans la zone de liste déroulante Police et en déterminant un corps de caractères dans la zone Taille.
4. Pour masquer la barre des indicateurs en marge, décochez la case correspondante. Vous gagnerez de l’espace pour afficher le code, mais les indicateurs en marge ne seront plus affichés.
5. Cliquez sur OK pour valider les paramètres d’affichage du code définis.

Figure 5.16

L'onglet Format de l'éditeur permet de personnaliser l'affichage du code.



Les paramètres d'affichage du code tels qu'ils sont définis par défaut dans Visual Basic Editor assurent une lecture confortable à l'écran. Si vous souhaitez cependant personnaliser les paramètres d'affichage du code, veillez à conserver cette qualité de lecture.

Déplacer une procédure

Si vous êtes un adepte des grands ménages à l'arrivée du printemps, il vous arrivera de souhaiter mettre de l'ordre dans vos projets VBA. Vous serez alors probablement amené à déplacer des procédures d'un module à un autre, afin de structurer de façon cohérente vos projets. Pour déplacer une procédure vers un autre module, utilisez la technique du glisser-déplacer :

1. Ouvrez les fenêtres Code concernées et affichez-les simultanément à l'aide des commandes Mosaique horizontale ou Mosaique verticale du menu Fenêtre.
2. Sélectionnez la procédure que vous souhaitez déplacer.
3. Cliquez sur le texte ainsi sélectionné et, tout en maintenant le bouton de la souris enfoncé, déplacez le curseur vers la fenêtre Code voulue. Une barre verticale grise indique où la procédure sélectionnée sera déplacée.
4. Relâchez le bouton de la souris.



S'il est impossible d'effectuer un glisser-déplacer dans la fenêtre Code, sélectionnez Outils puis Options et cochez la case Glisser-déplacer pour l'édition de texte de l'onglet Éditeur.



N'oubliez pas que l'Enregistreur de macro peut réduire de façon considérable le travail d'écriture de code. N'hésitez pas à enregistrer les tâches qui peuvent l'être dans une macro. Ajoutez ensuite les instructions Visual Basic qui ne peuvent être enregistrées dans la fenêtre Code de la macro.

Si vous devez ajouter des instructions à une macro, vous pouvez enregistrer les commandes correspondantes dans une nouvelle macro, puis coller le code ainsi généré à l'emplacement voulu.

Appel et sortie d'une procédure

La division d'un programme en plusieurs modules effectuant des tâches précises en améliore la lisibilité et facilite les éventuels débogages. Pour exécuter le programme, l'utilisateur exécute une procédure. Celle-ci effectue alors les tâches pour lesquelles elle a été définie et peut appeler une autre procédure qui prendra en charge l'exécution d'autres tâches. Elle pourra elle-même appeler d'autres procédures, et ainsi de suite. Lorsqu'une procédure en appelle une autre, cette dernière s'exécute, puis la procédure appelante reprend la main et se poursuit avec l'instruction qui suit l'instruction d'appel.

Appel d'une procédure *Sub*

Appeler une procédure consiste à demander à une procédure de s'exécuter à partir d'une autre procédure. La procédure appelée est exécutée, puis la procédure appelante reprend la main. Pour appeler une procédure de type Sub, utilisez le mot clé `Call`, selon la syntaxe suivante :

```
Call NomProcédure
```

où *NomProcédure* est le nom de la procédure à appeler.



Une instruction contenant simplement le nom d'une procédure suffit à appeler cette dernière. L'omission du mot clé `Call` est cependant déconseillée, car le code y perd en lisibilité.

Considérez l'exemple suivant :

```
Sub AppelsDeProcédures()  
    MsgBox "1er message de la procédure 1"  
    Call Procédure2  
    MsgBox "2e message de la procédure 1"  
End Sub
```

```
Sub Procédure2()  
    MsgBox "1er message de la procédure 2"  
    Call Procédure3  
    MsgBox "2e message de la procédure 2"  
End Sub  
  
Sub Procédure3()  
    MsgBox "1er message de la procédure 3"  
    MsgBox "2e message de la procédure 3"  
End Sub
```

L'exécution de la procédure AppelsDeProcédures entraîne l'affichage successif des messages suivants :

- 1^{er} message de la procédure 1
- 1^{er} message de la procédure 2
- 1^{er} message de la procédure 3
- 2^e message de la procédure 3
- 2^e message de la procédure 2
- 2^e message de la procédure 1

La première instruction de la procédure AppelsDeProcédures affiche un message. La procédure Procédure2 est ensuite appelée à l'aide de l'instruction Call. Elle affiche un message, puis appelle à son tour la procédure Procédure3. Celle-ci s'exécute et affiche consécutivement deux messages. Elle se termine alors, et la procédure appelante, Procédure2, reprend la main. Elle se poursuit avec l'instruction placée immédiatement sous l'appel et affiche de nouveau un message. Elle se termine à son tour et rend la main à la procédure qui l'a appelée, AppelsDeProcédures, qui affiche un ultime message et se termine.



Pour appeler une procédure, celle-ci doit être visible pour la procédure appelante. Pour plus d'informations, reportez-vous à la section "La notion de portée", plus haut dans ce chapitre.

Appels de procédures *Function* et *Property*

Les procédures Function et Property ont pour première fonction de renvoyer une valeur – vous pouvez y placer des instructions effectuant des tâches précises dans un document, mais il est préférable de réserver ces opérations aux procédures Sub. Pour appeler une procédure Function ou Property, il suffit de placer le nom de la procédure dans une

expression à l'emplacement où une valeur est attendue. La procédure Property ou Function s'exécute alors et se voit affecter une valeur qui se substitue à son appel dans la procédure appelante.

Pour des exemples d'appels de ce type de procédures, reportez-vous aux sections "Procédures *Function*" et "Procédures *Property*", plus haut dans ce chapitre.

Passage d'arguments

Lorsqu'une procédure est appelée, il est souvent nécessaire de lui passer des valeurs issues de la procédure appelante. La procédure appelée exploite alors les valeurs qui lui sont transmises et peut à son tour passer des valeurs à une autre procédure. Les arguments que peut recevoir une procédure d'une procédure appelante doivent apparaître entre parenthèses dans l'instruction de déclaration de la procédure. Ces arguments peuvent être facultatifs ou obligatoires, de type défini ou non.

L'argument passé à la procédure appelée peut être une valeur ou toute expression renvoyant une valeur acceptée par l'argument, c'est-à-dire d'un même type de données – les types de données sont étudiés au Chapitre 6.

Pour une revue détaillée de la syntaxe de déclaration des arguments d'une procédure, reportez-vous au Tableau 5.1.

Passage d'arguments par ordre d'apparition

Pour passer des arguments à une procédure, vous pouvez faire apparaître les valeurs à passer entre parenthèses dans l'instruction d'appel, en séparant chaque argument transmis par une virgule et en respectant l'ordre d'apparition des arguments dans la déclaration de la procédure appelée. Plus concrètement, pour passer des arguments à la procédure suivante :

```
Sub MaProcédure(Arg1, Arg2, Arg3)
```

utilisez une instruction Call de la façon suivante :

```
Call MaProcédure(ValArg1, ValArg2, ValArg3)
```

où *ValArg1*, *ValArg2* et *ValArg3* sont respectivement les valeurs que prendront les arguments Arg1, Arg2 et Arg3 de la procédure MaProcédure.

Certains arguments sont optionnels ; la procédure appelée peut alors s'exécuter correctement sans que ces arguments lui soient passés par la procédure appelante. Pour ignorer un argument lors d'un appel de procédures, placez deux virgules consécutives dans l'instruction d'appel, correspondant à l'emplacement de l'argument dans l'instruction de déclaration

de la procédure appelée. Par exemple, l'instruction utilisée pour passer les arguments Arg1 et Arg3 à la procédure MaProcédure sera :

```
Call MaProcédure(ValArg1, , ValArg3)
```

Lorsque vous ignorez un argument, veillez à ne pas oublier de placer une virgule pour cet argument dans l'instruction d'appel de procédure. Un tel oubli générerait un décalage dans le passage de valeur à la procédure appelée. L'instruction suivante :

```
Call MaProcédure(ValArg1, ValArg3)
```

passerait respectivement les valeurs ValArg1 et ValArg3 aux arguments Arg1 et Arg2 de la procédure MaProcédure.



Pour vérifier si des arguments facultatifs ont été omis lors de l'appel de la procédure, utilisez la fonction IsMissing selon la syntaxe suivante :

```
IsMissing(NomArgument)
```

La valeur True est renvoyée si aucune valeur correspondant à l'argument n'a été passée à la procédure ; False dans le cas contraire.



La fonction IsMissing fonctionne avec des valeurs de type Variant. Si vous testez un argument d'un autre type, IsMissing renverra toujours False, que l'argument ait été passé ou non.

Arguments nommés

Le passage d'arguments à une procédure selon leur ordre d'apparition dans l'instruction de déclaration de la procédure appelée est parfois périlleux. En particulier lorsque des arguments facultatifs ne sont pas passés par la procédure appelante ; une simple virgule oubliée dans l'instruction d'appel peut alors générer une erreur du programme ou, pire, des résultats erronés dont il sera difficile de détecter la source.

L'utilisation des *arguments nommés* permet d'éviter de tels problèmes, en nommant la valeur passée à la procédure afin d'éviter toute ambiguïté. Les arguments nommés sont des arguments dont le nom est reconnu par le programme. Lorsque vous déclarez une procédure, les arguments qui lui sont affectés deviennent des arguments nommés. Au lieu de passer les arguments selon leur ordre d'apparition, vous pouvez affecter des valeurs aux arguments nommés, dans l'ordre de votre choix, en mentionnant le nom de l'argument dans l'instruction d'appel, selon la syntaxe suivante :

```
Call NomProcédure (ArgNommé:= valeur, AutreArgNommé:= valeur)
```

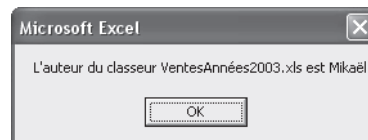
où *NomProcédure* est le nom de la procédure appelée et *valeur*, les valeurs affectées aux arguments nommés. Dans l'exemple suivant, la procédure *PassageArgumentsNommés* appelle *MaProcédure* et lui passe les arguments nommés *NomFichier* et *Propriétaire* :

```
Sub PassageArgumentsNommés()  
    Call MaProcédure(Propriétaire:=ActiveWorkbook.BuiltinDocumentProperties(3), _  
                    NomFichier:=ActiveWorkbook.Name)  
End Sub  
  
Sub MaProcédure(NomFichier As String, Propriétaire As String)  
    MsgBox "L'auteur du classeur" & NomFichier & "est" & _  
        Propriétaire  
End Sub
```

La procédure *MaProcédure* accepte les arguments *NomFichier* et *Propriétaire*, tous deux de type *String* (chaîne de caractères). La procédure *PassageArgumentsNommés* lui passe ces arguments, sans tenir compte de leur ordre d'apparition, mais en utilisant leurs noms d'arguments nommés. L'argument *Propriétaire* reçoit pour valeur la chaîne renvoyée par l'expression *ActiveWorkbook.BuiltinDocumentProperties(3)*. La propriété *BuiltinDocumentProperties* renvoie une collection *DocumentProperties* qui représente toutes les propriétés de document prédéfinies. L'index 3 permet de ne renvoyer que le troisième membre de la collection, en l'occurrence le nom de l'auteur. *NomFichier* se voit affecter la chaîne retournée par l'expression *ActiveWorkbook.Name* qui renvoie le nom du classeur actif. Ces arguments sont exploités par la procédure appelée pour afficher la boîte de dialogue représentée à la Figure 5.17.

Figure 5.17

Utilisez les noms des arguments nommés pour passer des informations à une procédure dans un ordre aléatoire.



Les arguments nommés ne sont pas une spécificité des procédures. La plupart des fonctions intégrées de Visual Basic intègrent des arguments nommés.

Sortie d'une procédure

Il peut être utile de quitter une procédure avant la fin de son exécution. Pour quitter une procédure avant la fin de son exécution, utilisez le mot clé *Exit* suivi du mot clé déterminant le type de la procédure :

- **Exit Sub.** Cette procédure entraîne la sortie d'une procédure de type Sub.
- **Exit Property.** Elle entraîne la sortie d'une procédure de type Property.
- **Exit Function.** Elle entraîne la sortie d'une procédure de type Function.

Lorsqu'une sortie de procédure est provoquée par le mot clé Exit, le programme se poursuit avec l'instruction suivant immédiatement celle qui a appelé la procédure quittée. Dans l'exemple suivant, une boîte de dialogue s'affiche à l'aide de la fonction MsgBox, afin de demander à l'utilisateur s'il souhaite exécuter de nouveau le programme.

```
Sub MonProgramme()  
    Instructions  
    Call AutreProcédure  
    Instructions  
    Recommencer = MsgBox "Recommencer l'opération ?", vbYesNo + vbQuestion  
    If Recommencer = vbYes Then  
        Call MonProgramme  
    Else  
        Exit Sub  
    End If  
End Sub
```

La procédure MonProgramme exécute des instructions, puis appelle une autre procédure. Elle reprend ensuite la main et exécute une autre série d'instructions. Une boîte de dialogue s'affiche, proposant à l'utilisateur de réitérer l'opération. Une structure de contrôle If...End If est utilisée pour déterminer le comportement du programme, en fonction de la réponse de l'utilisateur. S'il choisit le bouton Oui, la procédure s'appelle elle-même et s'exécute de nouveau ; s'il choisit Non, une instruction Exit entraîne la sortie de la procédure et le programme se termine.



Les structures de contrôle et les fonctions MsgBox et InputBox sont étudiées au Chapitre 7.

Sortie d'un programme

Deux instructions permettent d'interrompre l'exécution d'un programme : End et Stop.

Une instruction End met fin à l'exécution d'un programme et libère l'ensemble des ressources mémoire qu'il utilise. Les variables perdent leur valeur, les feuilles UserForm sont déchargées, et les éventuels classeurs ouverts par le programme à l'aide de la méthode Open sont fermés.

Si vous votre programme utilise des variables objet ou charge des feuilles UserForm, l'instruction End permet de réinitialiser l'ensemble des valeurs, vous préservant ainsi d'éventuels problèmes de mémoire. D'autre part, si le programme est exécuté à nouveau lors de la session Excel en cours, toutes les valeurs seront réinitialisées comme lors de la première exécution. Si vous ne mettez pas fin au programme par une instruction End et si vous ne libérez pas les ressources utilisées par le programme par des instructions appropriées, les valeurs telles que les données entrées dans une feuille UserForm restent chargées en mémoire. Si la feuille est à nouveau affichée au cours de session Excel, elle le sera avec ses valeurs.

L'instruction Stop est utilisée dans le cadre du débogage (voir Chapitre 10) et de l'analyse des programmes VBA afin de placer l'exécution d'un programme en mode Arrêt. Lorsqu'un programme atteint une instruction Stop, il s'interrompt et Visual Basic Editor s'ouvre sur la fenêtre Code du module contenant l'instruction Stop qui apparaît en surbrillance. Les ressources mémoire ne sont pas libérées et les variables conservent leurs valeurs. Vous pouvez poursuivre l'exécution pas à pas, écrire des instructions dans la fenêtre Exécution afin d'évaluer ou de tester votre programme ou encore poursuivre l'exécution du programme. Les outils et les techniques de débogage des programmes VBA constituent le sujet du Chapitre 10.

Exécuter du code



Pour exécuter le code à partir de Visual Basic Editor, placez le curseur dans la procédure que vous souhaitez exécuter, puis cliquez sur le bouton Exécuter de la barre d'outils Standard, ou choisissez la commande Exécuter Sub/UserForm du menu Exécution, ou, enfin, appuyez sur la touche F5.

Seules les procédures Sub ne nécessitant pas d'arguments sont accessibles par la boîte de dialogue Macro d'Excel. Pour exécuter une procédure Sub à partir de l'application hôte, choisissez Outils > Macro > Macros. Dans la boîte de dialogue Macros, sélectionnez la procédure à exécuter, puis cliquez sur le bouton Exécuter.



Macro or not macro ? Les macros sont des procédures de type Sub pouvant être exécutées de façon autonome, et donc ne nécessitant pas d'arguments. Une procédure qui attend des arguments ne peut qu'être appelée par une autre procédure et n'apparaît pas dans la liste des macros de la boîte de dialogue Macro.

Vous pouvez aussi affecter un bouton de barre d'outils ou un menu de commande à une procédure Sub. L'activation de ce bouton ou de ce menu exécute alors la procédure. Pour plus de précisions, reportez-vous au Chapitre 11.



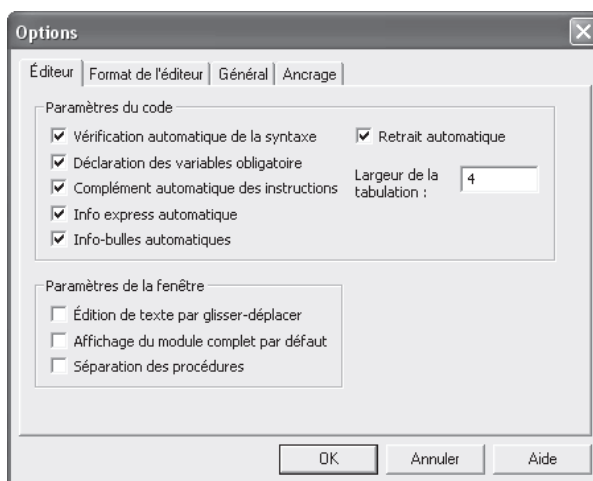
Si l'exécution d'un programme génère une erreur, reportez-vous au Chapitre 10.

Aide à l'écriture de code

Visual Basic Editor met à votre disposition des outils d'aide à l'écriture de code dans Visual Basic Editor. Pour activer ou désactiver ces outils, choisissez la commande Options du menu Outils et activez l'onglet Éditeur. Cochez ou décochez ensuite les options de la zone Paramètres du code (voir Figure 5.18). Si vous débutez dans la programmation en VBA, activez les options d'aide à l'écriture de code. Ils vous accompagneront dans votre apprentissage.

Figure 5.18

Visual Basic Editor propose des outils d'aide à l'écriture de code.



Vérification automatique de la syntaxe

L'option de vérification automatique de la syntaxe entraîne la vérification automatique de la validité de chaque ligne de code saisie. Chaque fois que vous frappez la touche Entrée ou que vous changez de ligne, la ligne en cours est vérifiée. Si une erreur est détectée, un message décrivant l'erreur de syntaxe s'affiche, et l'instruction non valide apparaît en rouge (voir Figure 5.19).

Vous pouvez alors cliquer sur le bouton OK et corriger l'erreur si elle vous apparaît évidente, ou choisir le bouton Aide afin d'afficher la rubrique d'aide associée à l'erreur repérée. Celle-ci vous présente les sources probables de l'erreur et vous propose des solutions adaptées (voir Figure 5.20). Dès que vous entrez une modification dans l'instruction incriminée,

celle-ci retrouve sa couleur normale, que l'erreur soit ou non corrigée. Si l'erreur de syntaxe n'est pas résolue, le message s'affichera de nouveau lorsque vous changerez de ligne.

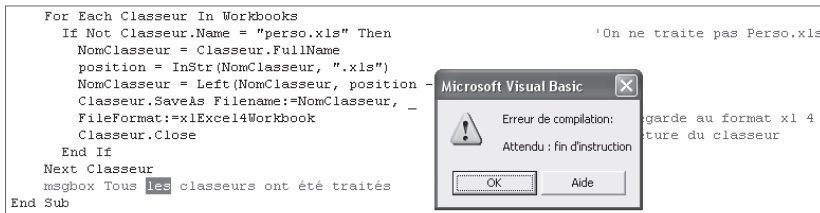
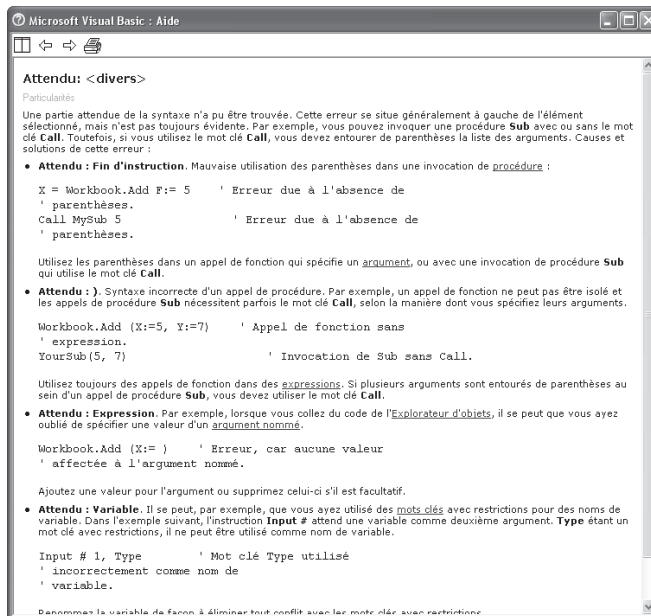


Figure 5.19

Vous êtes prévenu chaque fois qu'une erreur est détectée.

Figure 5.20

Si vous ne parvenez pas à décoder l'origine de l'erreur, activez la rubrique d'aide Visual Basic.



Complément automatique des instructions

L'option Complément automatique des instructions affiche une liste alphabétique de mots clés possibles chaque fois que l'attente d'un complément est reconnue lors de l'écriture de code. C'est par exemple le cas lorsque vous saisissez un nom de propriété appelant un objet, directement suivi d'un point. Visual Basic Editor reconnaît alors qu'un membre de l'objet (propriété ou méthode) est attendu et en affiche la liste (voir Figure 5.21).

Vous pouvez alors sélectionner l'un des éléments de la liste à l'aide de la souris ou des touches fléchées puis valider ce choix par la touche Espace, ou continuer à saisir votre code, sans tenir compte de la liste affichée.

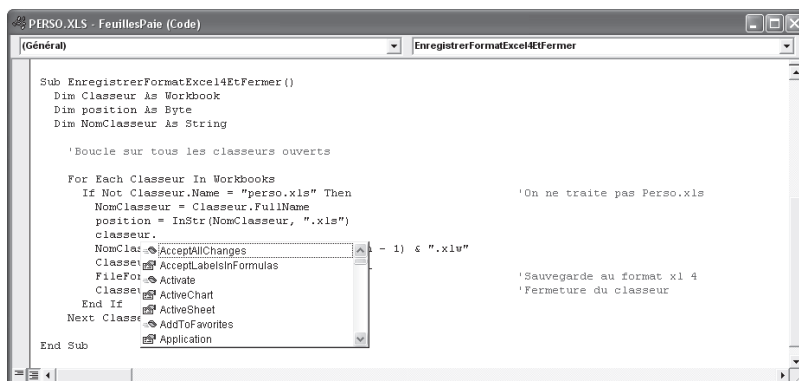


Figure 5.21

L'option de complément automatique des instructions affiche la liste des mots clés possibles.

Info express automatique

Chaque fois qu'une fonction intégrée de Visual Basic ou qu'une procédure Fonction du module est reconnue, l'option Info express automatique en affiche la syntaxe détaillée, dans un cadre situé sous l'instruction saisie (voir Figure 5.22).

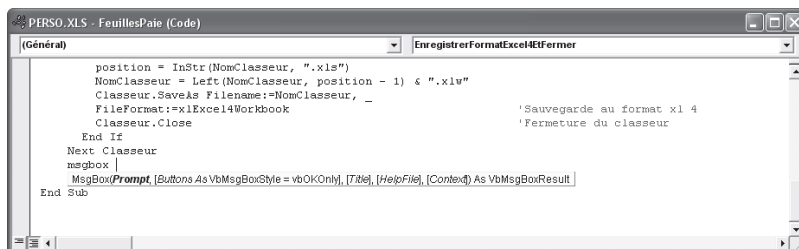


Figure 5.22

L'option d'info express automatique affiche la syntaxe des fonctions au cours de la saisie.



Les options d'aide à l'écriture de code dans Visual Basic Editor peuvent aussi être activées par la barre d'outils Édition.



6

Variables et constantes

Au sommaire de ce chapitre

- Déclarer une variable
- Types de données des variables
- Portée et durée de vie des variables
- Traitement interapplications à l'aide de variables objet

Les variables sont un élément essentiel de la programmation. Elles permettent de stocker les informations de votre choix à tout moment de l'exécution d'un programme et de les réexploiter à n'importe quel autre moment. Les variables sont un lieu de stockage que vous déterminez. Vous pouvez, par exemple, stocker dans une variable une valeur représentant le nombre de classeurs ouverts, le nom du fichier, la valeur ou l'adresse d'une cellule, les informations entrées par l'utilisateur dans une feuille VBA, etc.

Déclarer une variable

Pour créer une variable, vous devez la déclarer, c'est-à-dire lui affecter un nom qu'il suffira par la suite de réutiliser pour exploiter la valeur qui y est stockée. La déclaration de variables dans un programme VBA peut être implicite ou explicite. Autrement dit, les programmes VBA peuvent reconnaître l'utilisation d'une nouvelle variable sans que celle-ci soit préalablement créée dans une instruction de déclaration. Vous pouvez aussi paramétrer Visual Basic Editor afin d'exiger la déclaration explicite des variables avant leur utilisation.

Déclaration implicite

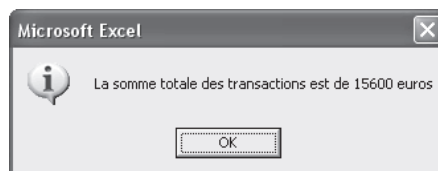
Si la déclaration explicite des variables n'est pas requise, le simple fait de faire apparaître un mot non reconnu par le programme dans une instruction d'affectation suffira pour que ce nom soit considéré comme une variable de type Variant par le programme – les types de variables sont présentés plus loin dans ce chapitre. C'est le cas dans l'exemple suivant :

```
Sub DéclarImpliciteDeVariables()  
    mavar = Range("D7").Value  
    MsgBox "La somme totale des transactions est" & mavar, _  
        vbInformation + vbOKOnly  
    Instructions  
End Sub
```

Le mot MaVar apparaît pour la première fois dans une instruction d'affectation valide. La variable MaVar est donc créée et affectée à la valeur de la cellule D7. Elle est ensuite utilisée pour afficher un message à l'attention de l'utilisateur (voir Figure 6.1). L'opérateur de concaténation & est utilisé pour faire apparaître la valeur de la variable au cœur d'une chaîne de caractères définie.

Figure 6.1

Les variables peuvent être utilisées dans des chaînes à l'aide de l'opérateur de concaténation.



Vous pouvez attribuer le nom de votre choix à une variable. Un nom de variable doit cependant respecter les règles suivantes :

- Il doit commencer par une lettre.
- Il ne peut contenir plus de 255 caractères.
- Le point, le point d'exclamation, les espaces et les caractères @, &, \$ et # ne sont pas autorisés.
- Ce nom ne doit pas être un mot réservé, c'est-à-dire un mot reconnu comme un élément du langage Visual Basic (nom de fonction, d'objets, de propriété, d'argument nommé, etc.).

Déclaration explicite

Si les variables peuvent être déclarées implicitement, il est fortement recommandé de déclarer explicitement les variables avant de les utiliser.

Forcer la déclaration des variables avec *Option Explicit*

Visual Basic Editor permet de forcer la déclaration des variables avant leur utilisation, grâce à l'instruction `Option Explicit`. L'utilisation de cette option permet d'éviter les risques d'erreurs liées à une faute de frappe dans le nom d'une variable. Considérez la procédure suivante :

```
Sub MacroErreur()  
    MaVariable = Workbooks.Count  
    While MaVariable < 10  
        Workbooks.Add  
        MaVariable = MaVariable + 1  
    Wend  
End Sub
```

Cette procédure a pour but d'ouvrir des classeurs en boucle, jusqu'à ce que dix classeurs soient ouverts au total. Le nombre de classeurs ouverts (`Workbooks.Count`) est affecté à la variable `MaVariable`. Une instruction `While...Wend` – que vous découvrirez dans le prochain chapitre – est utilisée pour créer un nouveau classeur (`Workbooks.Add`) et ajouter 1 à `MaVariable` tant que la valeur de celle-ci est inférieure à 10. Cependant, le nom de la variable a été incorrectement saisi dans l'instruction suivante :

```
While MaVariable < 10
```

"`MaVariable`" n'existant pas, cette variable est créée et aucune valeur ne lui est affectée. La procédure ouvre donc des documents et incrémente `MaVariable` de 1 à l'infini, sans que la condition `MaVariable < 10` ne soit jamais respectée.



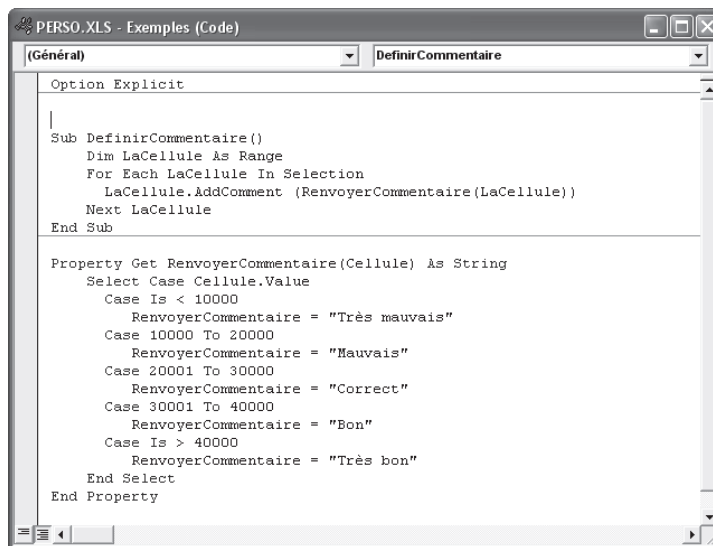
Pour interrompre une macro s'exécutant à l'infini, tapez la combinaison clavier Ctrl+Pause.

Pour éviter ce type d'erreur, forcez la déclaration explicite des variables. Pour cela, placez-vous dans la section Déclarations de la fenêtre Code du module et saisissez-y l'instruction `Option Explicit` (voir Figure 6.2).

Figure 6.2

L'instruction

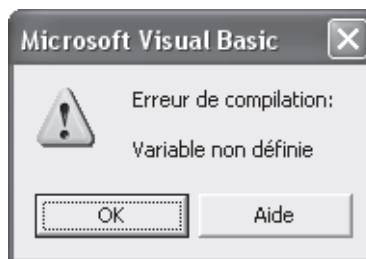
Option Explicit doit se trouver dans la section Déclarations de la fenêtre Code.



Dorénavant, l'apparition de noms de variables non préalablement déclarées à l'aide de l'instruction `Dim` dans une procédure générera une erreur. Dans notre exemple, l'utilisation de l'instruction `Option Explicit` entraînera le message d'erreur présenté à la Figure 6.3 à l'exécution du programme.

Figure 6.3

L'option Option Explicit permet de détecter immédiatement les erreurs de saisie dans les noms de variables.



Vous pouvez aussi paramétrer Visual Basic Editor pour que l'instruction `Option Explicit` soit systématiquement placée dans la section de Déclarations de tout nouveau module.

1. Sélectionnez la commande Options du menu Outils et placez-vous sur l'onglet Éditeur.
2. Cochez la case Déclaration explicite des variables, puis cliquez sur OK.

L'instruction `Option Explicit` sera désormais automatiquement placée dans la section appropriée de la fenêtre Code de chaque nouveau module.

Déclarer une variable avec *Dim*

La déclaration de variables se fait à l'aide de l'instruction `Dim`, selon la syntaxe suivante :

```
Dim NomVariable As Type
```

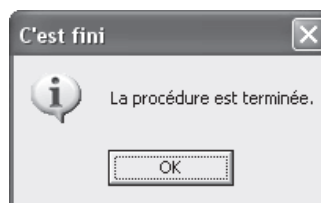
où *NomVariable* est le nom de la variable, qui sera utilisé par la suite pour y faire référence, et *Type* le type de données qu'accepte la variable – présentés dans la section suivante. L'argument *Type* est facultatif, mais la déclaration d'un type de variable peut vous faire économiser de l'espace mémoire et améliorer ainsi les performances de votre programme. Lors de la déclaration de variables, une variable de type `String` prend pour valeur une chaîne vide, une variable numérique prend la valeur 0.

Dans l'exemple suivant, les variables `Message`, `Boutons` et `Titre` sont déclarées à l'aide de l'instruction `Dim`, des valeurs leur sont ensuite affectées, puis sont utilisées comme arguments de `MsgBox` afin d'afficher la boîte de dialogue de la Figure 6.4.

```
Sub UtiliserDim()  
    Dim Message As String  
    Dim Boutons As Single  
    Dim Titre As String  
    Message = "La procédure est terminée."  
    Boutons = vbOKOnly + vbInformation  
    Titre = "C'est fini"  
    MsgBox Message, Boutons, Titre  
End Sub
```

Figure 6.4

Des variables peuvent être utilisées comme arguments d'une fonction.



Plusieurs variables peuvent être déclarées dans une même instruction `Dim`, selon la syntaxe suivante :

```
Dim NomVar1 As Type, NomVar2 As Type, ..., NomVarN As Type
```


où *NomVar1* est le nom de la première variable déclarée, et ainsi de suite. Les trois instructions de déclaration de l'exemple suivant peuvent ainsi être ramenées à une seule instruction de déclaration :

```
Dim Message As String, Boutons As Single, Titre As String
```

Gardez à l'esprit que pour affecter un type aux variables d'une telle instruction, celui-ci doit être mentionné pour chacune des variables déclarées. L'instruction suivante déclare une variable *Message* de type *Variant* et une variable *Titre* de type *String* :

```
Dim Message, Titre As String
```

Types de données des variables

Lorsque vous créez une variable, vous pouvez déterminer son *type*. Le type d'une variable correspond au type de l'information qui peut y être stockée. Il peut s'agir d'une valeur numérique (un nombre ou une expression renvoyant un nombre), d'une chaîne de caractères, d'une date, etc. Les valeurs affectées aux variables dans des instructions d'affectation doivent être compatibles avec le type de la variable. Par exemple, déclarer une variable de type numérique et lui affecter par la suite une chaîne de caractères générera une erreur.

Vous pouvez aussi choisir de ne pas déterminer le type d'une variable. Elle sera alors de type *Variant*, et vous pourrez y stocker tous les types de données. Cette section présente les différents types de variables.

Chaînes de caractères

Les variables de type *String* – encore appelées "variables de chaîne" – permettent le stockage de données reconnues comme une chaîne de caractères. Pour déclarer une variable de type *String*, utilisez la syntaxe suivante :

```
Dim NomVariable As String
```

Une variable de chaîne peut être affectée à toute expression renvoyant une valeur de type chaîne (chaîne, propriété, fonction, etc.). Pour affecter une chaîne de caractères à une variable de type *String*, placez celle-ci entre guillemets. Si vous souhaitez placer des guillemets dans une chaîne de caractères (ou tout autre caractère), utilisez conjointement l'opérateur de concaténation & et la fonction *Chr* selon la syntaxe suivante :

```
"Chaîne de car." & Chr(codeANSI) & "Chaîne de car."
```

où *codeANSI* est le code ANSI du caractère que vous souhaitez insérer.



L'opérateur + peut aussi être utilisé pour concaténer des chaînes de caractères. Préférez cependant l'opérateur &, afin de distinguer aisément la concaténation de chaînes d'additions de valeurs numériques qui, elles, requièrent l'opérateur +.

Les instructions d'affectation suivantes sont toutes valides :

- **Prénom = "Luc"**
- **NomFichier = ActiveWorkbook.Name**
- **Message = "Le nom du classeur actif est " & Chr(34) & ActiveWorkbook.Name & Chr(34) & "."**

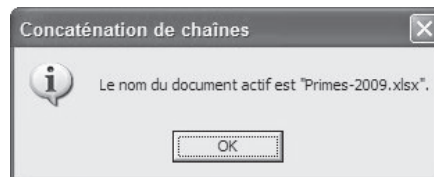
La première instruction affecte une chaîne définie à la variable Prénom. La seconde instruction affecte la valeur de la propriété Name du classeur actif à la variable NomFichier. La troisième conjugue l'affectation de chaînes définies dans le texte, renvoyées par une fonction et renvoyées par une propriété, en concaténant ces différentes chaînes à l'aide de l'opérateur &.

La macro suivante affiche la boîte de dialogue représentée à la Figure 6.5.

```
Sub ConcatenerLesChaines()
    Dim Message As String, Boutons As Single, Titre As String
    Message = "Le nom du document actif est" & Chr(34) & _
        ActiveWorkbook.Name & Chr(34) & "."
    Boutons = vbInformation + vbOKOnly
    Titre = "Concaténation de chaînes"
    MsgBox Message, Boutons, Titre
End Sub
```

Figure 6.5

Le message affiché par la fonction MsgBox est toujours une chaîne de caractères.



Les variables de chaîne définies précédemment sont dites de *longueur variable* et peuvent être constituées d'environ deux milliards de caractères. Vous pouvez cependant déclarer des variables de chaîne de *longueur fixe*, autorisant de 1 à 65 400 caractères, selon la syntaxe suivante :

```
Dim NomVariable As String * longueur
```

où *longueur* est le nombre de caractères que peut recevoir la variable. La déclaration de variables de chaîne de longueur fixe peut générer une économie de mémoire – et donc un gain de performance – pour votre programme, mais doit être utilisée prudemment. En effet, si la chaîne affectée à une variable de longueur fixe dépasse la capacité de cette dernière, la chaîne sera purement et simplement rognée. Remplacez l’instruction de déclaration des variables de l’exemple précédent par celle-ci :

```
Dim Message As String * 15, Boutons As Single, Titre As String * 5
```

Vous obtenez la boîte de dialogue présentée à la Figure 6.6.

Figure 6.6
Utilisez les variables de chaîne de longueur fixe avec prudence.



Valeurs numériques

Les variables numériques permettent le stockage de valeurs exploitables en tant que telles, c’est-à-dire sur lesquelles vous pouvez effectuer des opérations arithmétiques. Il existe plusieurs types de variables numériques. Elles se distinguent par l’échelle des valeurs qu’elles acceptent et par la place qu’elles occupent en mémoire. Le Tableau 6.1 présente les différents types de variables numériques :

Tableau 6.1 : Types de données numériques

<i>Types de données</i>	<i>Valeurs acceptées</i>	<i>Mémoire occupée</i>
Byte (octet)	Nombre entier, compris entre 0 et 255	1 octet
Integer (entier)	Nombre entier compris entre –32 768 et 32 767	2 octets
Long (entier long)	Nombre entier compris entre –2 147 483 648 et 2 147 483 647	4 octets
Single (simple précision)	Nombre à virgule flottante compris entre –1,401298E-45 et –3,402823E38 pour les nombres négatifs, entre 1,401298E-45 et 3,402823E38 pour les nombres positifs	4 octets

Types de données	Valeurs acceptées	Mémoire occupée
Double (double précision)	Nombre à virgule flottante compris entre -1,79769373486232E308 et -4,94065645841247E-324 pour les nombres négatifs, entre 4,94065645841247E-324 et 1,79769373486232E308 pour les nombres positifs	8 octets
Currency (monétaire)	Nombre à virgule fixe, avec quinze chiffres pour la partie entière et quatre chiffres pour la partie décimale, compris entre -922 337 203 685 477,5808 et 922 337 203 685 477,5807	8 octets

Si votre programme exploite beaucoup de variables, l’affectation du type approprié – celui qui exploite le moins d’espace mémoire – aux différentes variables en améliorera les performances. Veillez cependant à ce que les valeurs qu’une variable est susceptible de prendre soient toujours couvertes par le type de données acceptées par la variable. Par exemple, si une valeur supérieure à 255 ou inférieure à 0 est affectée à une variable de type Byte, une erreur sera générée (voir Figure 6.7).

Figure 6.7
Le type de données acceptées par une variable doit couvrir l’ensemble des valeurs possibles lors de l’exécution du programme.



Utilisez le point comme séparateur décimal dans le code VBA. L’utilisation de la virgule génère une erreur.

Au même titre que les variables de type String, une variable numérique peut être concaténée avec une chaîne de caractères à l’aide de l’opérateur &. Une variable numérique peut aussi être affectée à toute expression renvoyant une valeur numérique (chaîne, propriété, fonction, etc.). Vous pouvez aussi effectuer des opérations arithmétiques sur les variables numériques en utilisant les opérateurs arithmétiques présentés dans le Tableau 6.2.

Tableau 6.2 : Les opérateurs arithmétiques

<i>Opérateur</i>	<i>Description</i>
+	Addition.
-	Soustraction.
*	Multiplication.
/	Division.
\	Division. Seule la partie entière du résultat de la division est renvoyée (l'opération 18 \ 5 retournera la valeur 3).
^	Élévation à la puissance (2 ^ 4 renvoie 16).

À condition que le type de la variable numérique MaVar couvre les valeurs qui lui sont affectées, les instructions d'affectation suivantes sont toutes valides :

- **MaValeur = 58**
- **NbreClasseur = Workbooks.Count**
- **Range("D5").Value = (Range("D3").Value + Range("D4").Value) / 2**
- **SurfaceCercle = (varRayon ^ 2) * 3.14**

La première instruction affecte une valeur définie à la variable MaValeur. La deuxième instruction affecte la valeur de la propriété Count de l'objet (la collection) Workbooks (le nombre de classeurs ouverts). La troisième utilise l'opérateur arithmétique + pour additionner les valeurs des cellules D3 et D4, et l'opérateur arithmétique / pour diviser la valeur obtenue par deux. La dernière instruction élève la variable varRayon précédemment définie au carré à l'aide de l'opérateur ^, puis multiplie cette valeur par 2 à l'aide de l'opérateur *.

Par contre, l'instruction suivante affecte une valeur de type chaîne à la variable numérique Mavar et génère une erreur (voir Figure 6.8).

```
Sub ErreurAffectation()
    Dim Mavar As Byte
    MaVar = ActiveWorkbook.Name
End Sub
```

Rappelez-vous que les constantes sont en réalité des valeurs numériques. Une constante peut donc être affectée à une variable numérique et utilisée dans une expression arithmétique. Veillez à ne pas utiliser l'opérateur &, réservé à la concaténation de chaînes, dans une instruction d'affectation de valeur à une variable numérique. Par exemple, l'instruction :

```
MsgBox "Le message de la bdg", vbOKOnly + vbInformation, "Titre"
```

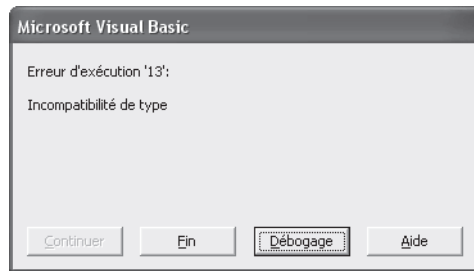
est valide, tandis que l'instruction :

```
MsgBox "Le message de la bdg", vbOKOnly & vbInformation, "Titre"
```

générera un message d'erreur, car l'opérateur & est utilisé pour additionner les valeurs affectées aux constantes vbOKOnly et vbInformation.

Figure 6.8

Une chaîne de caractères ne peut être affectée à une variable numérique.



VBA intègre de nombreuses fonctions permettant de manipuler les valeurs numériques. Les deux fonctions suivantes en sont des exemples :

- **ABS(*nombre*)**. Renvoie la valeur absolue du nombre spécifié entre parenthèses.
- **Int(*nombre*)**. Renvoie la partie entière du nombre spécifié entre parenthèses.

La liste des fonctions VBA est consultable dans l'aide en ligne. Choisissez la rubrique Manuel de référence du langage. Vous y trouverez une rubrique Fonctions répertoriant les fonctions par ordre alphabétique. Vous pouvez également y choisir la rubrique Liste et index. Vous y découvrirez des rubriques thématiques, telles que Résumé des mots clés financiers, ou Résumé des mots clés mathématiques.

Valeurs booléennes

Les variables de type Boolean servent à stocker le résultat d'expressions logiques. Une variable de type Boolean peut renvoyer la valeur True ou False et occupe deux octets en mémoire. Une variable booléenne peut recevoir la valeur retournée par une expression renvoyant une valeur de type Boolean, mais peut aussi être affectée à une valeur numérique. Dans ce cas, si la valeur numérique est égale à zéro, la variable prendra la valeur False, True dans le cas contraire. Une variable booléenne utilisée comme chaîne de caractères renvoie le mot Vrai ou Faux.

Dans l'exemple suivant, la variable booléenne ClasseurSauvegardé prend la valeur False si le classeur actif a subi des modifications depuis le dernier enregistrement ; True dans le cas contraire. Une boîte de dialogue affiche ensuite le message Vrai ou Faux, en fonction de la valeur de ClasseurSauvegardé.

```
Sub ClasseurSauvegardéOuNon()  
    Dim ClasseurSauvegardé As Boolean  
    ClasseurSauvegardé = ActiveWorkbook.Saved  
    MsgBox ClasseurSauvegardé  
End Sub
```

Dates

Les dates comprises entre le 1^{er} janvier 100 et le 31 décembre 9999 peuvent être affectées à des variables de type Date. Ces variables sont stockées sous la forme de nombres à virgule flottante et occupent huit octets en mémoire.

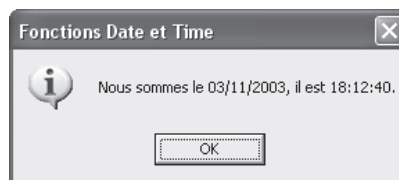
La partie entière du nombre représente le jour. 0 correspond au 30 décembre 1899, 1 au 31 décembre 1899, etc. Les valeurs négatives représentent des dates antérieures au 30 décembre 1899. La partie décimale du nombre représente l'heure. 0.5 correspond à 12 heures, 0.75, à 18 heures, etc. Ainsi, 36526.00001 correspond au 1^{er} janvier 2000, à 00:00:01.

Visual Basic intègre des fonctions permettant de manipuler les dates. Par exemple, les fonctions Date, Time et Now renvoient respectivement la date courante, l'heure courante et la date courante suivie l'heure courante. La procédure suivante affiche la boîte de dialogue de la Figure 6.9 :

```
Sub DateEtHeure()  
    Dim LaDate As Date, LHeure As Date  
    LaDate = Date  
    LHeure = Time  
    MsgBox "Nous sommes le" & LaDate & ", il est "& LHeure & ".", _  
        vbOKOnly + vbInformation, "Fonctions Date et Time"  
End Sub
```

Figure 6.9

Visual Basic intègre des fonctions permettant de manipuler les dates et les heures.



Pour plus d'informations sur les fonctions de date et d'heure, reportez-vous à l'aide Visual Basic, en choisissant Résumé des mots clés de date et d'heure de la rubrique Index/Listes du Manuel de référence du langage. Vous pouvez également vous reporter au Manuel de référence pour Microsoft Excel, disponible dans le sommaire de l'aide.



L'installation par défaut d'Excel n'installe pas l'aide en ligne de VBA. Si vous n'avez pas effectué une installation personnalisée, vous devrez lancer de nouveau l'installation et installer l'aide en ligne de VBA.

Type Variant

Une variable, ou une variable de type Variant, accepte des valeurs de tout type et peut être automatiquement convertie d'un type à l'autre, tant que sa valeur est compatible avec le type vers lequel s'opère la conversion. Autrement dit, une variable de type Variant peut être initialement une chaîne de caractères, et être exploitée par la suite en tant que valeur numérique. Si les données qui lui sont affectées sont assimilables à une valeur numérique, la conversion se fera automatiquement, lorsque l'instruction assimilable à une opération numérique sera exécutée.

Si les variables de type Variant sont très pratiques, elles occupent un espace en mémoire plus important que les autres types de variables (16 octets pour les valeurs numériques et 22 octets pour les valeurs de chaîne) et peuvent donc ralentir l'exécution du programme.

Une variable, une constante ou un argument dont le type n'est pas déclaré sont assimilés à une variable de type Variant. Vous pouvez cependant spécifier le type Variant pour faciliter la lecture de votre code. Les instructions suivantes sont équivalentes :

```
Dim MaVar  
Dim MaVar As Variant
```

Variables de matrice

Une variable de matrice ou de type Array, parfois appelée tableau, est une variable permettant de stocker plusieurs valeurs. Contrairement à une variable ordinaire ne pouvant recevoir qu'une seule valeur, vous pouvez déterminer plusieurs emplacements de stockage pour une variable de matrice. Les variables de matrice permettent de stocker des listes d'éléments de même type dans une même variable. Vous pouvez par exemple stocker dans une variable de matrice les chiffres d'affaires de tous les représentants. Pour déclarer une variable de matrice, utilisez la syntaxe suivante :

```
Dim NomVariable(NbreElements) As Type
```

où *NomVariable* est le nom de la variable, *NbreElements*, le nombre de valeurs que recevra la variable, et *Type*, le type de données qu'accepte la variable. Pour affecter des valeurs à une variable de matrice ou accéder à celles-ci, il suffit de spécifier la position de la valeur stockée dans la variable. La première valeur recevant l'index 0, la dernière valeur est toujours égale au nombre d'éléments contenus dans la variable moins 1.



Pour démarrer l'index d'un tableau à 1 plutôt qu'à 0, placez l'instruction `Option Base 1` dans la section Déclarations du module.

Une variable de matrice peut également être déclarée selon la syntaxe suivante :

```
Dim NomVariable(Début To Fin) As Type
```

où *Début* et *Fin* définissent la plage de valeurs qui sera utilisée pour stocker et accéder aux données de la variable.

Une variable de matrice peut vous servir à stocker des données de même type. Il est alors recommandé de déclarer un type approprié. Dans l'exemple suivant, la variable de matrice `JoursSemaine` stocke sous forme de chaînes les jours de la semaine. Une structure de contrôle `For...Next` est ensuite utilisée pour afficher les valeurs contenues par la variable dans une boîte de dialogue.

```
Sub VarMatrice()
    Dim JoursSemaine(7) As String
    JoursSemaine(0) = "Lundi"
    JoursSemaine(1) = "Mardi"
    JoursSemaine(2) = "Mercredi"
    JoursSemaine(3) = "Jeudi"
    JoursSemaine(4) = "Vendredi"
    JoursSemaine(5) = "Samedi"
    JoursSemaine(6) = "Dimanche"
    Dim compteur As Byte
    For compteur = 0 To 6
        MsgBox JoursSemaine(compteur)
    Next compteur
End Sub
```

Une variable de matrice peut aussi servir à stocker des données de types différents. Elle doit alors être de type `Variant`. La procédure suivante stocke dans une seule variable le nom, la date de naissance, l'adresse, la fonction et le salaire d'un employé. Ces données sont ensuite affichées dans une boîte de dialogue (voir Figure 6.10).

```
Sub InfosEmployé
    Dim Employé(1 To 5) As Variant
    Employé(1) = "Jean Dupont"
    Employé(2) = "25/12/71"
    Employé(3) = "14, rue des Arts"
    Employé(4) = "Chargé d'études"
    Employé(5) = 12000
```

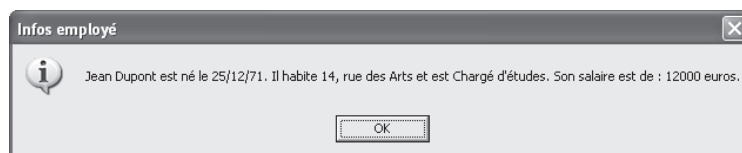
```

MsgBox Employé(1) & " est né le " & Employé(2) & ". Il habite " & _
Employé(3) & " et est " & Employé(4) & ". Son salaire est de : " & _
Employé(5) & " euros", vbOKOnly + vbInformation, "Infos employé"
End Sub

```

Figure 6.10

Toutes les informations relatives à un même sujet peuvent être stockées dans une seule variable de matrice.



Utilisez la fonction *IsArray* pour vérifier si une variable est de type Array.

Variables de matrice multidimensionnelles

Les variables de matrice peuvent être multidimensionnelles. Les données sont alors stockées horizontalement et verticalement, à la manière d'un tableau. Les variables de matrice multidimensionnelles permettent de stocker de façon cohérente les valeurs d'une feuille Excel. Les variables multidimensionnelles se déclarent selon la même syntaxe que les variables de matrice à une dimension, en ajoutant simplement les arguments d'index de début et de fin pour la deuxième dimension :

```
Dim NomVariable(Début1 To Fin1, Début2 To Fin2) As Type
```

Considérez la feuille de classeur représentée à la Figure 6.11. Elle représente les ventes mensuelles pour l'année 2003, de quatre types de produits, soit 12 lignes sur quatre colonnes. Une variable de matrice multidimensionnelle permettra de stocker l'ensemble des valeurs de cette feuille logiquement.

Figure 6.11

Utilisez les variables de matrice multidimensionnelles pour stocker les valeurs d'un tableau.

	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00	54 897,00	
6	Mai	45 258,00	32 568,00	36 550,00	32 568,00	
7	Juin	38 652,00	97 632,00	56 320,00	97 632,00	
8	Juillet	32 510,00	45 863,00	45 520,00	45 863,00	
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	
14						

La déclaration de la variable de matrice multidimensionnelle se présentera comme suit :

```
Dim MonTableau(1 To 12, 1 To 4) As Single
```

Cette variable correspond à un tableau de données de 12 lignes sur 4 colonnes. Il suffit ensuite d'affecter logiquement les valeurs de la feuille aux espaces de stockage de la variable :

```
MonTableau(1, 1) = Cells(3, 2).Value  
MonTableau(1, 2) = Cells(3, 3).Value  
MonTableau(1, 3) = Cells(3, 4).Value  
MonTableau(1, 4) = Cells(3, 5).Value  
MonTableau(2, 1) = Cells(4, 2).Value  
MonTableau(2, 2) = Cells(4, 3).Value  
MonTableau(2, 3) = Cells(4, 4).Value  
MonTableau(2, 4) = Cells(4, 5).Value  
Etc.
```

Ainsi, pour accéder aux ventes d'un mois, il suffira de spécifier la valeur correspondante comme premier index de la variable MonTableau (1 = janvier, 2 = février, etc.). Pour accéder à une catégorie de ventes, il suffira de spécifier la valeur correspondante comme second index de la variable MonTableau (1 = Livres, 2 = Vidéo, 3 = Hi-Fi, 4 = Autres). Par exemple, MonTableau(1, 1) renverra le chiffre des ventes du mois de janvier pour les livres ; MonTableau(12, 2) renverra le chiffre des ventes du mois de décembre pour la vidéo.

N'hésitez pas à utiliser les variables de matrice pour stocker les données d'une feuille Excel auxquelles un programme VBA doit accéder à de multiples reprises. La variable ainsi créée est chargée en mémoire. L'accès aux données qu'elle contient en est sensiblement plus rapide qu'un accès aux valeurs contenues dans les cellules d'une feuille de calcul.

L'utilisation d'une structure de contrôle For...Next permettra d'affecter l'ensemble des valeurs de ce tableau à une variable de matrice en quelques lignes de code. Vous apprendrez à utiliser cette structure au Chapitre 7.

Info

Une variable de matrice n'est pas limitée à deux dimensions. Vous pouvez parfaitement créer une variable de matrice à trois dimensions, ou plus.

Conseil

Les fonctions LBound et UBound renvoient respectivement le plus petit indice et le plus grand indice disponible pour une dimension spécifiée d'un tableau et s'utilisent selon la syntaxe suivante :

`LBound(NomVariable, Dimension)` et `UBound(NomVariable, Dimension)`

Où NomVariable est le nom de la variable de matrice et Dimension, la dimension dont vous souhaitez connaître le plus petit ou le plus grand indice (1 pour la première dimension, 2 pour la deuxième, etc.). Si l'argument Dimension est omis, le plus petit ou le plus grand indice de la première dimension est renvoyé.

Variables de matrice dynamiques

Les variables de matrice dynamiques sont des variables de matrice dont vous pouvez modifier la taille. Pour déclarer une variable de matrice dynamique, ne spécifiez pas de valeur entre les parenthèses qui suivent son nom. L'instruction suivante :

```
Dim NomVariable()
```

crée une variable de matrice dynamique. Avant d'affecter des valeurs à la variable de matrice ainsi créée, vous devrez la redimensionner à l'aide de l'instruction ReDim, selon la syntaxe suivante :

```
ReDim NomVariable(Début To Fin)
```

Vous pouvez utiliser le mot clé ReDim pour redimensionner une variable de matrice autant de fois que vous le souhaitez. Les variables de matrice sont intéressantes lorsque vous ne connaissez pas la quantité de données à stocker. Supposez que, dans l'exemple précédent, la feuille de calcul des ventes ne soit pas une feuille de calcul annuelle, mais mensuelle. Le tableau s'enrichirait alors tous les mois d'une nouvelle ligne. Pour que votre programme fonctionne tout au long de l'année, vous devrez créer une variable de matrice de longueur variable. C'est ce que fait la procédure suivante :

```
1: Sub AffectationVariableArray()  
2:   Dim MonTableau() As Single  
3:   Dim DerniereLigne As Byte  
4:   DerniereLigne = Range("A3").End(xlDown).Row  
5:   Dim NbreDeLignes As Byte  
6:   NbreDeLignes = DerniereLigne - 2  
7:   ReDim MonTableau(NbreDeLignes, 4)  
8:   Instructions d'affectation de valeurs à MonTableau  
9: End Sub
```

Aux lignes 2 et 3, les variables MonTableau et DerniereLigne sont déclarées. L'instruction de la ligne 4 sert à affecter le numéro de la dernière ligne contenant des données à la variable DerniereLigne. La fonction End renvoie l'objet Range correspondant à la dernière cellule non vide sous (xlDown) la cellule A3. La propriété Row renvoie la valeur du numéro de ligne de cet objet. La variable NbreDeLignes est créée ligne 5. On lui affecte ensuite une valeur égale à DerniereLigne - 2, soit le nombre de lignes contenant des données à

stocker dans la variable (les deux premières lignes ne contenant pas de données à stocker). À la ligne 7, la variable de matrice `MonTableau` est redimensionnée de façon à accueillir l'ensemble des chiffres de ventes de la feuille.



Lorsque vous redimensionnez une variable de matrice, celle-ci est réinitialisée et toutes les valeurs qui y étaient stockées sont perdues. Pour conserver les valeurs d'une variable de matrice lors de son redimensionnement, placez le mot clé `Preserve` devant l'instruction `ReDim`. L'utilisation du mot clé `Preserve` est cependant subordonnée à certaines conditions :

- Vous ne pouvez redimensionner que la dernière dimension de la variable.
- Vous ne pouvez pas modifier le nombre de dimensions du tableau.
 - Vous ne pouvez qu'agrandir la taille du tableau. Si vous réduisez la taille de la variable, toutes les données seront perdues.

La fonction première d'Excel étant d'effectuer des calculs sur des données affichées sous forme de tableaux, les variables de matrice sont très utilisées dans les programmes VBA pour Excel. Elles permettent en effet de stocker les données de feuilles de calcul sous forme de variables et de travailler directement sur la variable plutôt que sur le tableau, améliorant ainsi sensiblement les performances du programme.

Variables objet

Les variables objet sont utilisées pour faire référence à un objet et occupent 4 octets en mémoire. Une fois une variable objet définie, vous pouvez interroger ou définir les propriétés de l'objet, exécuter l'une de ses méthodes en faisant simplement référence à la variable. Pour déclarer une variable objet, utilisez la syntaxe suivante :

```
Dim NomVariable As Object
```

Vous pouvez déclarer précisément le type d'objet affecté à la variable, en remplaçant `Object` par un nom d'objet reconnu par l'application. Vous pouvez, par exemple, déclarer une variable objet `Workbook` (classeur) selon la syntaxe suivante :

```
Dim MonObjetClasseur As Workbook
```

Une fois la variable déclarée, vous devez lui affecter un objet précis, utilisez pour cela le mot clé `Set`, selon la syntaxe suivante :

```
Set NomVariable = Expression
```

où *Expression* est une expression renvoyant un objet de l'application. Dans l'exemple suivant, la variable objet `Police` est déclarée en tant qu'objet `Font`, puis affectée à

l'expression `Workbooks("Representant.xlsx").Sheets("Feuil1").Range("A1:D5").Font`, soit l'objet `Font` (police) de la plage de cellules A1:D5 de la feuille libellée "Feuil1" du classeur `Representant.xlsx`. La variable objet est ensuite utilisée pour définir la propriété `Bold` de l'objet à `True`, c'est-à-dire pour affecter l'attribut gras à la plage de cellules A1:D5 de ce classeur.

```
Sub VariablesObjet()  
    Dim Police As Font  
    Set Police = _  
        Workbooks("Representant.xlsx").Sheets("Feuil1").Range("A1:D5").Font  
    Police.Bold = True  
End Sub
```

La fonction *GetObject*

Les variables objet vous permettent d'agir sur un objet sans que celui-ci soit ouvert. Vous pouvez ainsi interroger les valeurs d'un Tableau Excel sans que celui-ci soit ouvert. Vous pouvez également modifier l'objet auquel vous accédez ainsi. Pour accéder à un objet, stockez cet objet dans une variable objet et à l'aide de la fonction `GetObject`, selon la syntaxe suivante :

```
Set MonObjet = GetObject(pathname, class)
```

où *pathname* et *class* sont des arguments nommés de type chaîne, correspondant respectivement au chemin d'accès complet et au nom du fichier auquel vous souhaitez accéder et à la classe de l'objet. Si l'argument *pathname* est spécifié, l'argument *class* peut être omis.

La procédure suivante crée une variable objet de type `Workbook` et lui affecte le fichier `Representant.xlsx`, situé sur le Bureau de Windows.

```
Sub AccederObjetFerme()  
    Dim ObjetClasseur As Workbook  
    Set ObjetClasseur = _  
        GetObject("C:\Windows\Bureau\Representant.xlsx")  
End Sub
```

La fonction `GetObject` est particulièrement intéressante si des données entrées dans un classeur doivent être répercutées dans un ou plusieurs autres classeurs. Vous pouvez ainsi créer un programme VBA, afin que, lorsqu'une commande est effectuée, le classeur contenant les données de stock soit mis à jour. Si nécessaire, un message pourra être affiché, afin de prévenir l'utilisateur qu'il est temps de renouveler le stock, et ce sans même qu'il sache qu'il existe un classeur des stocks.

C'est ce que fait la procédure suivante, en supposant que la valeur du stock pour le produit commandé se trouve dans la cellule A13 du classeur stock.xlsx.

```

1: Sub Commande()
2:   'Instructions Dim UnitésCommandées As Integer
3:   Dim StockRestant As Integer
4:   Dim UnitésCommandées As Integer
5:   UnitésCommandées = 50
6:   StockRestant = VerifierEtMettreAJourStock(UnitésCommandées)
7:   If StockRestant < 0 Then
8:     MsgBox "Le stock ne permet pas d'assurer la commande. " & _
       "Le stock pour ce produit est de " & _
       (StockRestant + UnitésCommandées) & " unités."
9:     Exit Sub
10:  Else
11:    MsgBox "Commande effectuée. Le stock restant pour ce " & _
      "produit est de " & StockRestant & " unités."
12:  End If
13:  'Suite des instructions de la commande
14: End Sub

15: Function VerifierEtMettreAJourStock(QteCommande)
16:   Dim ObjetStock As Workbook
17:   Dim StockDispo As Integer
18:   Set ObjetStock = GetObject("C:\Documents and Settings\Administrateur\Bureau\Stock.xlsx")
19:   StockDispo = ObjetStock.Sheets(1).Range("A13").Value
20:   VerifierEtMettreAJourStock = StockDispo - QteCommande
21:   If VerifierEtMettreAJourStock >= 0 Then
22:     ObjetStock.Sheets(1).Range("A13").Value = _
       StockDispo - QteCommande
23:     ObjetStock.Save
24:   End If
25: End Function

```



Veillez à personnaliser le chemin précisé pour la fonction GetObject à la ligne 18, sinon cette macro ne fonctionnera pas.

À la ligne 6, la procédure Commande appelle la procédure VerifierEtMettreAJourStock en lui passant la valeur de la variable UnitésCommandées. La valeur 50 a été affectée à cette variable à la ligne 5 pour faire fonctionner le programme. Il va de soi que cette variable doit être affectée au nombre d'unités réellement commandées.

La fonction `VerifierEtMettreAJourStock` déclare la variable objet `ObjetStock` de type `Workbook` (ligne 16) et la variable `StockDispo` de type `Integer` (ligne 17). Ligne 18, la variable `ObjetStock` est affectée au classeur `Stock.xlsx` situé sur le Bureau de Windows. À la ligne suivante, la variable `StockDispo` reçoit la valeur de la cellule A13 de la première feuille de ce classeur. La fonction `VerifierEtMettreAJourStock` se voit ensuite affecter la valeur de `StockDispo - QteCommande`. Enfin, lignes 21 à 24, une structure `If...Then...Else` est utilisée pour mettre à jour la valeur du stock restant. L'instruction de la ligne 21 vérifie que le stock restant après commande est supérieur à zéro. Si c'est le cas, la valeur de la cellule A13 est mise à jour pour refléter le stock restant (ligne 22) et le classeur est ensuite sauvegardé (ligne 23).

La procédure `Commande` reprend alors la main. Lignes 7 à 12, une structure conditionnelle `If...Then...Else` affiche un message à l'attention de l'utilisateur, afin de l'informer sur l'état du stock. Si le stock est insuffisant pour assurer la commande (`StockRestant < 0`), l'instruction de la ligne 8 est exécutée. L'utilisateur est alors averti que la commande n'a pu être validée, et informé du stock disponible (`StockRestant + UnitésCommandées`). Si le stock permet d'assurer la commande, l'instruction de la ligne 11 affiche un message informant l'utilisateur du stock restant après commande.

La fonction *CreateObject*

La fonction `CreateObject` sert à créer une instance d'un objet, et s'utilise selon la syntaxe suivante :

```
CreateObject(class,servername)
```

où `class` et `servername` sont des arguments nommés de type chaîne. `class` correspond à la classe de l'objet dont on crée une instance. `servername` est facultatif et correspond au nom d'un serveur distant sur lequel est créé l'objet. Si vous omettez cet argument, l'objet est créé sur la machine sur laquelle s'exécute le programme.

Une fois créée une instance d'objet, on accède aux propriétés et aux méthodes de cet objet en utilisant le nom de la variable à laquelle il est affecté.

Dans l'exemple suivant, une instance de l'objet Excel est créée. Un nouveau classeur est créé, configuré puis enregistré dans cette instance.

```
1: Sub CreerInstancesExcel()  
2:   'déclaration des variables  
3:   Dim Xl As Excel.Application  
4:   Dim NouvClasseur As Excel.Workbook  
5:   Dim NomFichier As String  
6:   'création d'une instance de l'objet Excel  
7:   Set Xl = CreateObject("Excel.Application")
```



```
8: 'affichage de l'objet Xl
9: Xl.Application.Visible = True
10: 'création d'un nouveau classeur dans l'objet Xl
11: Set NouvClasseur = Xl.Workbooks.Add
12: 'suppression de la troisième feuille de calcul
13: NouvClasseur.Sheets(3).Delete
14: 'affectation de noms aux feuilles 1 et 2
15: NouvClasseur.Sheets(1).Name = "Quantites"
16: NouvClasseur.Sheets(2).Name = "Chiffres"
17: 'définition du nom du classeur
18: Dim compteur As Byte
19: Dim Pos As Long
20: NomFichier = "Ventes " & Date & ".xlsx"
21: For compteur = 1 To 2
22:   Pos = InStr(NomFichier, "/")
23:   NomFichier = Left(NomFichier, Pos - 1) & "-" & _
24:     Right(NomFichier, Len(NomFichier) - Pos)
25: Next compteur
26: 'enregistrement du classeur
27: NouvClasseur.SaveAs "C:\Documents and Settings\Administrateur\Bureau\
   Ventes\" & NomFichier
28: NouvClasseur.Close
29: Xl.Quit
30: End Sub
```



Veillez à personnaliser le chemin précisé pour la fonction `GetObject` à la ligne 27, sinon cette macro ne fonctionnera pas.

Lignes 2 à 5, les variables sont déclarées. Les variables `Xl` et `NouvClasseur` sont des variables objet de type `Excel` et `Workbook`. La variable `NomFichier` servira à stocker le nom d'enregistrement du classeur. Ligne 7, une instance de l'objet `Application` d'Excel est créée à l'aide de l'instruction `CreateObject`, et affectée à la variable `Xl`. La propriété `Visible` de l'objet `Application` est ensuite définie à `True` afin de faire apparaître la session Excel à l'écran.

Ligne 11, un nouveau classeur est ajouté à l'objet `Application` nouvellement créé, et affecté à la variable objet `NouvClasseur`. Notez que, par défaut, un nouveau classeur est créé dans la session Excel à partir de laquelle le programme est exécuté. Pour que le nouveau classeur soit créé dans la nouvelle session Excel, il est indispensable de faire référence à l'objet `Xl` dans l'instruction de la ligne 11 (`Xl.Workbooks.Add` et pas simplement `Workbooks.Add`). Lignes 13 à 16, la troisième feuille du classeur est supprimée et les deux autres sont renommées.

Lignes 17 à 25, le nom d'enregistrement du classeur est défini. La variable `NomFichier` se voit tout d'abord affecter le nom `Ventes` suivi de la date du jour (`Ventes 12/08/2007`, par exemple). Ce nom contient deux fois le caractère barre oblique (/), non valide dans les noms de fichier. Lignes 21 à 25, une boucle `For...Next` est utilisée pour répéter deux fois le traitement appliqué au nom du classeur afin de substituer des traits d'union aux barres obliques (les boucles sont étudiées au prochain chapitre). On utilise pour cela les fonctions de manipulation de chaîne `InStr`, `Left`, `Right` et `Len`. `InStr` est utilisé pour renvoyer la position du caractère / dans la chaîne `NomFichier` (ligne 22). `Len` renvoie le nombre de caractères de la chaîne `NomFichier` (ligne 24). Les fonctions `Left` et `Right` sont utilisées pour renvoyer respectivement les caractères situés à gauche et à droite des barres obliques, et un trait d'union est placé entre les deux chaînes ainsi renvoyées.

Lignes 27 et 28, le classeur est enregistré puis fermé. Ligne 29, la méthode `Quit` est appliquée à l'objet `Excel`, afin de fermer la session `Excel` créée en début de programme.

Libérer une variable objet

Pour annuler l'affectation d'un objet à une variable objet, donnez-lui la valeur `Nothing`. L'instruction suivante annule l'affectation de la variable objet `Police` créée précédemment :

```
Set Police = Nothing
```

Il est important d'affecter la valeur `Nothing` à une variable objet, lorsque celle-ci n'est plus utilisée par le programme. Vous libérez ainsi l'ensemble des ressources système et mémoire associées à l'objet. Dans l'exemple de programme précédent, vous devrez placer cette instruction au-dessus de la ligne 24. La variable objet `ObjetStock` sera ainsi libérée avant que la procédure `Commandes` ne reprenne la main.

Types de données personnalisés

Le mot clé `Type` permet de créer des types de données personnalisés, associant les types de données présentés ci-dessus. Une telle opération se révèle intéressante lorsqu'un programme doit associer de façon récurrente différents types de données. Vous pouvez alors créer un nouveau type de données auquel il suffira de faire référence chaque fois que vous souhaitez créer une nouvelle variable regroupant ces types de données.

À l'instar des variables de matrice, le mot clé `Type` permet donc de créer des variables capables de stocker des informations multiples. Mais, contrairement aux variables de matrice, `Type` permet d'associer des types de données différents. La déclaration d'un nouveau type de données doit être placée dans la section `Déclarations` du module, selon la syntaxe suivante :

```
Type NomType
    Données1 As Type
    Données2 As Type
    ....
    Donnéesn As Type
End Type
```

où *Données1*, ..., *Donnéesn* sont les données que contiendront les variables de type *NomType*. Ces noms seront employés par la suite pour affecter des valeurs aux différents espaces de stockage des variables de type *NomType*. *Type* représente le type de données affecté à chaque élément du nouveau type de données.

Dans l'exemple suivant, un type de données *Membre* est créé, afin de pouvoir intégrer dans une seule variable l'ensemble des informations concernant un membre d'une association :

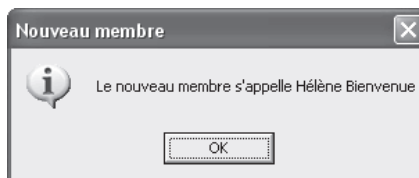
```
Type Membre
    Prénom As String
    Nom As String
    Adresse As String
    CodePostal As String
    Ville As String
    Téléphone As String
    Age As Byte
End Type
```

Vous pouvez maintenant créer une nouvelle variable de type *Membre*, qui sera prête à recevoir toutes les informations contenues dans ce type. Les informations contenues dans une variable peuvent ensuite être interrogées ou définies en faisant suivre le nom de la variable d'un point, puis du nom de la donnée. La procédure suivante crée une variable de type *Membre* et définit ses valeurs, puis affiche une boîte de dialogue indiquant les données *Prénom* et *Nom* de la variable (voir Figure 6.12) :

```
Sub NouveauMembre
    Dim NouvMembre As Membre
    With NouvMembre
        .Prénom = "Hélène"
        .Nom = "Bienvenue"
        .Adresse = "4, rue des oiseaux"
        .CodePostal = "56000"
        .Ville = "Vannes"
        .Téléphone = "00 01 02 03 04"
        .Age = 2
    End With
    MsgBox "Le nouveau membre s'appelle " & NouvMembre.Prénom & " " & _
        NouvMembre.Nom, vbOKOnly + vbInformation, "Nouveau membre"
End Sub
```

Figure 6.12

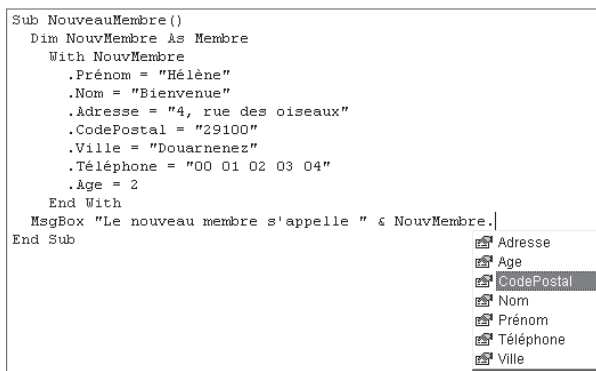
Les variables de type personnalisé peuvent contenir un nombre indéterminé d'informations.



Notez que, en phase de création, un complément automatique s'affiche lorsque vous faites référence à une variable de type personnalisé (voir Figure 6.13) – à condition que l'option correspondante soit validée.

Figure 6.13

Le complément automatique d'instruction s'affiche pour les types de données personnalisés.



Constantes

Les constantes permettent d'attribuer un nom à une valeur fixe. Il est ainsi plus aisé d'exploiter cette valeur dans le code en faisant référence au nom de la constante, plutôt qu'à la valeur elle-même. Par ailleurs, si une valeur est susceptible d'être modifiée (une valeur de TVA, par exemple), l'affectation de cette valeur à une constante simplifiera les éventuelles mises à jour. Il vous suffira en effet de modifier la valeur de la constante, plutôt que de modifier chaque occurrence de la valeur dans le code de l'ensemble de vos projets.



Une fois qu'une valeur a été affectée à une constante, celle-ci ne peut être modifiée par la suite.

Pour déclarer une constante, utilisez l'instruction `Const`, selon la syntaxe suivante :

```
Const NomConstante As Type = Valeur
```

où *NomConstante* est le nom de la constante, *Type*, le type de données de la constante – il peut s'agir de n'importe lequel des types de données présentés plus haut –, et *Valeur*, la

valeur qui lui est affectée. L'instruction suivante déclare la constante TVA, à laquelle la valeur 20.6 est affectée.

```
Const TVA As Single = 20.6
```

Validation et conversion des types de données

Il est souvent nécessaire de vérifier que le type de données entrées par l'utilisateur dans une cellule ou dans une feuille UserForm est valide, c'est-à-dire correspond au type de valeur attendue. Si tel n'est pas le cas, il est probable que le programme génère une erreur. Celle-ci peut alors être évitée en convertissant le type de la variable.

Vérifier le type de données d'une variable

VBA intègre des fonctions permettant de vérifier qu'une valeur correspond bien au type attendu. Ces fonctions sont présentées dans le Tableau 6.3.

Tableau 6.3 : Fonctions VBA permettant de vérifier les types de données

<i>Fonction</i>	<i>Description</i>
<code>IsArray(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> est une variable de matrice ; <code>False</code> dans le cas contraire.
<code>IsDate(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> est une variable de matrice ; <code>False</code> dans le cas contraire.
<code>IsNumeric(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> est un nombre ; <code>False</code> dans le cas contraire.
<code>IsObject(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> est une variable objet ; <code>False</code> dans le cas contraire.
<code>IsMissing(<i>MaVar</i>)</code>	Renvoie <code>True</code> si l'argument optionnel <i>MaVar</i> est de type <code>Variant</code> et n'a pas été passé à la fonction ou à la procédure en cours.
<code>IsEmpty(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> a été initialisée, c'est-à-dire si une valeur lui a été affectée ; <code>False</code> dans le cas contraire. Valide uniquement pour les variables de type <code>Variant</code> .
<code>IsNull(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> contient la valeur <code>Null</code> ; <code>False</code> dans le cas contraire. Ne confondez pas une variable contenant une valeur <code>Null</code> et une variable qui n'a pas été initialisée et ne contient aucune valeur. Valide uniquement pour les variables de type <code>Variant</code> .
<code>IsError(<i>MaVar</i>)</code>	Renvoie <code>True</code> si la variable <i>MaVar</i> stocke une valeur qui correspond à l'un des codes d'erreur de VBA. <code>False</code> dans le cas contraire.

Vous pouvez également utiliser les fonctions `VarType` ou `TypeName` pour interroger le type d'une variable. Utilisez `VarType` selon la syntaxe suivante :

```
MaVar = VarType(NomVar)
```

où *MaVar* est une variable de type `Integer`, et *NomVar* le nom de la variable dont vous souhaitez interroger le type.

La variable *MaVar* reçoit pour valeur une constante Visual Basic, indiquant le type de la variable *NomVar* (`vbInteger` pour une variable de type `Integer`, `vbDate` pour une variable de type `Date`, etc.).

`TypeName` s'utilise selon la même syntaxe (*MaVar* = `VarType(NomVar)`), mais renvoie une chaîne de caractères représentant le type de la variable (voir Tableau 6.4).

Tableau 6.4 : Valeurs renvoyées par la fonction *TypeName*

<i>Chaîne renvoyée</i>	<i>Variable</i>
Type objet	Objet dont le type est <i>type_objet</i>
Byte	Octet
Integer	Entier
Long	Entier long
Single	Nombre à virgule flottante en simple précision
Double	Nombre à virgule flottante en double précision
Currency	Valeur monétaire
Decimal	Valeur décimale
Date	Valeur de date
String	Chaîne
Boolean	Valeur booléenne
Error	Valeur d'erreur
Empty	Non initialisée
Null	Aucune donnée valide
Object	Objet
Unknown	Objet dont le type est inconnu
Nothing	Variable objet qui ne fait pas référence à un objet

La fonction `EntrerUneDate` suivante utilise la fonction `InputBox` pour demander à l'utilisateur d'entrer une date. La fonction `IsDate` est employée pour vérifier si la valeur entrée par l'utilisateur est bien une date. Si tel n'est pas le cas, l'utilisateur est de nouveau invité à entrer une date.

```
Function EntrerUneDate()  
    Do  
        EntrerUneDate = InputBox("Entrez une date", "Vérification du type de  
        ➡ données")  
    Loop Until IsDate(EntrerUneDate) = True  
End Function
```



La fonction `InputBox` et la structure de contrôle `Do...Loop` sont présentées en détail au chapitre suivant.

Modifier le type d'une variable

VBA intègre des fonctions permettant de convertir une variable d'un type défini vers un autre type de données. Le Tableau 6.5 présente sommairement ces fonctions. Pour plus de précisions, reportez-vous à l'aide en ligne de VBA.

Tableau 6.5 : Fonctions de conversion de types de données

<i>Fonction</i>	<i>Description</i>
<code>CBool (MaVar)</code>	Convertit la variable. Renvoie <code>True</code> si <i>MaVar</i> est une valeur numérique différente de 0 ; <code>False</code> si <i>MaVar</i> est égale à 0. Une erreur est générée si <i>MaVar</i> n'est pas une valeur numérique.
<code>CByte (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Byte</code> ¹ .
<code>CCur (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Currency</code> (monétaire) ¹ .
<code>CDate (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Date</code> ¹ .
<code>CDbl (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Double</code> ¹ .
<code>CDec (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Decimal</code> ¹ .
<code>CInt (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Integer</code> ¹ .
<code>CLng (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Long</code> ¹ .
<code>CSng (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Single</code> ¹ .

<i>Fonction</i>	<i>Description</i>
<code>CVar (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>Variant</code> . <i>MaVar</i> doit être une valeur de type <code>Double</code> pour les nombres et de type <code>String</code> pour les chaînes.
<code>CStr (MaVar)</code>	Convertit <i>MaVar</i> en une variable de type <code>String</code> . Si <i>MaVar</i> est une valeur de type <code>Boolean</code> , la fonction <code>CStr</code> renvoie Vrai ou Faux. Si <i>MaVar</i> est une valeur de type <code>Date</code> , la fonction <code>CStr</code> renvoie la date sous forme de chaîne. Si <i>MaVar</i> est une valeur de type numérique, la fonction <code>CStr</code> renvoie cette valeur sous forme de chaîne.

1. *MaVar* doit être une valeur compatible avec le type de données vers lequel s'opère la conversion. Par exemple, si vous utilisez la fonction `Cbyte`, *MaVar* doit être une valeur numérique comprise entre 0 et 255. Si *MaVar* n'est pas une valeur compatible avec le type de données renvoyé par la fonction, une erreur "Type incompatible" est générée.

Notez que *MaVar* peut être une variable ou toute expression valide.

Portée et durée de vie des variables

Outre leur type et leur valeur, les variables et les constantes sont caractérisées par leur *portée*. La portée d'une variable ou d'une constante désigne son accessibilité pour les procédures et les modules du projet. Variables et constantes peuvent être accessibles à une procédure, à l'ensemble des procédures d'un module, ou encore à l'ensemble des modules du projet en cours. Les variables sont aussi caractérisées par leur *durée de vie*. La durée de vie d'une variable désigne le temps pendant lequel la variable conserve sa valeur. Une variable peut conserver sa valeur tant qu'une procédure s'exécute, et être réinitialisée lorsque la procédure est terminée, ou conserver sa valeur entre les différents appels de procédures.

Portée de niveau procédure

Une variable-constante est dite *de niveau procédure* lorsqu'elle n'est accessible qu'à la procédure dans laquelle elle est déclarée. C'est le cas de toute variable-constante déclarée à l'intérieur d'une procédure. Les variables-constantes déclarées à l'intérieur d'une procédure ne sont accessibles qu'à la procédure à laquelle elles appartiennent.

Portée de niveau module privée

Une variable-constante est dite *de niveau module privé* lorsqu'elle est accessible à l'ensemble des procédures du module dans lequel elle est déclarée. Elle doit pour cela être déclarée dans la section Déclarations du module, c'est-à-dire à l'extérieur de toute procédure.

Par défaut, les variables-constantes déclarées dans la section Déclarations d'un module ont une portée *privée*, c'est-à-dire ne sont accessibles qu'aux procédures du module. Vous pouvez cependant substituer le mot clé `Private` à `Dim` pour améliorer la lisibilité de votre code. La déclaration de la variable se présente alors ainsi :

```
Private NomVariable As Type
```

Dans l'exemple suivant, la constante `Pi` est déclarée de niveau module privé et est accessible à toutes les procédures du module.

```
Option Explicit
Private Pi As Single
Pi = 3.14

Sub Procédure-1 ()
[....] 'Instructions
End Sub

[....] 'Autres procédures du module

Sub Procédure-n ()
[....] 'Instructions
End Sub
```

Portée de niveau module publique

Une variable-constante est dite *de niveau module public* lorsqu'elle est accessible à l'ensemble des procédures du projet, quel que soit le module de stockage. Elle doit pour cela être déclarée dans la section Déclarations du module, à l'aide de l'instruction `Public`, selon la syntaxe suivante :

```
Public NomVariable As Type
```

Conseil

Lorsque vous utilisez une valeur définie de façon récurrente dans un projet (une valeur de TVA, par exemple), affectez-lui une constante de niveau module public et utilisez cette constante plutôt que la valeur elle-même dans les procédures. Si cette valeur est modifiée, il vous suffira de redéfinir l'instruction d'affectation de la constante pour mettre à jour la totalité du projet.

Variables statiques

Une variable conserve sa valeur tant que le programme s'exécute dans son champ de portée. Cette valeur peut être modifiée, mais la variable conserve une valeur. Lorsque l'exécution du programme sort de la portée de la variable, celle-ci est réinitialisée et perd sa valeur. Autrement dit, une variable de niveau procédure conserve sa valeur tant que la procédure dans laquelle elle est déclarée est en cours d'exécution – même lorsque la procédure appelle d'autres procédures. Lorsque celle-ci se termine, la variable est réinitialisée. Une variable de niveau module conserve sa valeur jusqu'à ce que le programme prenne fin.

Pour qu'une variable de niveau procédure conserve sa valeur entre différents appels, substituez le mot clé `Static` à `Dim` dans l'instruction de déclaration de la variable, selon la syntaxe suivante :

```
Static NomVariable As Type
```



Pour déclarer toutes les variables d'une procédure Sub ou Function statiques, placez le mot clé `Static` devant l'instruction de déclaration de la procédure.

Traitement interapplications à l'aide de variables objet

Une variable objet peut être affectée à un objet appartenant à une application autre que l'application hôte du projet. Un programme VBA Excel peut ainsi exploiter des objets du modèle d'objets Word, Access ou toute autre application supportant *Automation*. Il vous suffit pour cela d'affecter les objets auxquels vous souhaitez accéder à une variable objet à l'aide des fonctions `GetObject` et/ou `CreateObject`.



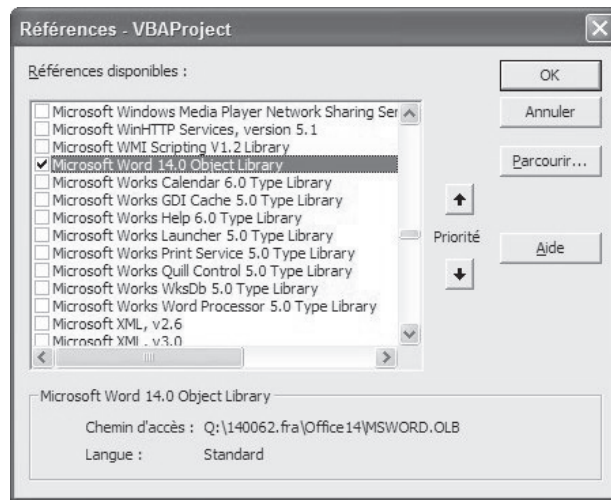
Automation, ou OLE Automation, est une fonction du modèle d'objets composant (COM, Component Object Model). Il s'agit d'un standard qu'utilisent les applications pour exposer leurs objets, méthodes et propriétés aux outils de développement. Les applications Office supportent Automation. Un classeur Excel peut ainsi exposer une feuille de calcul, un graphique, une cellule ou une plage de cellules, etc. Un fichier Word pourra exposer une page, un paragraphe, un mot, ou tout autre objet de son modèle d'objets. Visual Basic pour Applications permet d'accéder à ces objets, d'interroger ou de redéfinir leurs propriétés, d'en exécuter les méthodes, etc.

Pour qu'un projet puisse accéder à la bibliothèque d'objets d'une autre application, celle-ci doit être référencée dans le projet. Pour référencer une bibliothèque d'objets, choisissez la commande Référence du menu Outils. Dans la boîte de dialogue qui s'affiche, cochez les cases des bibliothèques que vous souhaitez référencer, puis cliquez sur OK.

Pour réaliser l'exemple suivant, référencez la bibliothèque d'objets Microsoft Word Object Library à partir d'un projet Excel. Puis créez un nouveau document Word et enregistrez-le sur le Bureau de Windows, sous le nom MonDoc.docx.

Figure 6.14

Activez la bibliothèque d'objets de l'application que vous souhaitez manipuler à partir de la boîte de dialogue Références.



Placez ensuite le code suivant dans un module Excel :

```

1: Sub InsereTableauDansFichierWord()
2:   Dim MonDoc As Object
3:   On Error Resume Next
4:   Set MonDoc = GetObject(, "Word.Application")
5:   If Err.Number <> 0 Then Err.Clear
6:   Set MonDoc = GetObject("C:\Documents and settings\Administrateur\Bureau\
   MonDocx.docx")
7:   Dim MaPosition As Word.Range
8:   Set MaPosition = MonDoc.Range(0, 0)
9:   MonDoc.Tables.Add Range:=MaPosition, NumRows:=3, NumColumns:=4
10:  MonDoc.Save
11:  Set MonDoc = Nothing
12:  Word.Application.Quit
13: End Sub

```



Veillez à personnaliser le chemin précisé pour la fonction `GetObject` à la ligne 18, sinon cette macro ne fonctionnera pas.

Exécutez la procédure, puis ouvrez le fichier Word. Un tableau de quatre colonnes sur trois lignes a été placé en début de document.

La variable objet `MonDoc` est déclarée ligne 2. Elle est ensuite affectée à l'objet `Word.Application`, représentant l'application Word, à l'aide de l'instruction `Set` et de la fonction `GetObject` (ligne 4). Si l'application Word n'est pas ouverte, une erreur est générée. Un détecteur d'erreurs est donc placé en ligne 3, de façon à ignorer l'erreur et à passer à l'instruction suivante. Si, effectivement, une erreur survient (`If Err.Number <> 0`), la propriété `Number` de l'objet `Err` est redéfinie à 0 (ligne 5). Cette éventuelle erreur étant gérée, `MonDoc` peut être affectée à `MonDoc.docx` (ligne 6).

La variable `MaPosition` de type `Word.Range` est ensuite déclarée (ligne 7) – l'objet `Range` de Word représente une position de curseur dans un document. Ligne 8, la position représentant le début du document lui est affectée. Un tableau est ensuite inséré à cette position, ligne 9. Le document est ensuite sauvegardé. Ligne 11, la variable `MonDoc` est libérée. Enfin, l'instruction de la ligne 12 permet de quitter Excel. En effet, lorsque vous faites appel à une variable objet d'une autre application, le moteur de l'application est lancé. N'omettez donc pas d'employer la méthode `Quit`, afin de libérer les ressources occupées par l'application.



On distingue, dans l'accès aux objets d'autres applications à l'aide d'Automation, la liaison tardive de la liaison précoce. La liaison est dite tardive lorsqu'une variable objet de type `Object` ou `Variant` est déclarée (`Dim MaVar As Object`). La variable est ensuite initialisée et affectée à un objet de l'application étrangère à l'aide de la fonction `GetObject`. On parle de liaison précoce lorsque la variable est déclarée d'un type identifiant l'application dont on souhaite exploiter les objets (`Dim MaVar as Word.Application`). Utilisez de préférence une liaison précoce dans vos programmes. Les performances du programme en seront améliorées, et Visual Basic vérifiera la syntaxe spécifique aux objets de l'application étrangère lors de l'écriture de votre programme.

Notez qu'un programme Excel peut exécuter une macro stockée dans une autre application hôte. Dans l'exemple suivant, la macro `MacroWord` est exécutée sur un nouveau document à partir d'Excel. Pour réaliser cet exemple, commencez par créer la macro `MacroWord` :

1. Lancez Word. Définissez le niveau de sécurité des macros de façon à autoriser l'exécution des macros (commande Sécurité des macros de l'onglet Développeur).



Le modèle Normal.dotm de Word est l'équivalent du classeur de macros personnel d'Excel : les macros stockées dans ce modèle sont accessibles à tous les documents Word. Dans les versions antérieures à Word 2007, ce modèle se nomme Normal.dot.

2. Activez l'onglet Développeur, puis cliquez sur la commande Macros. La boîte de dialogue Macros s'affiche.



Avec une version de Word antérieure à Word 2007, choisissez Outils > Macro > Macros.

3. Dans la zone Nom de la macro, saisissez MacroWord et, dans la liste déroulante Macros disponibles dans, sélectionnez Normal.dot (modèle global).



Le modèle Normal.dot de Word est l'équivalent du classeur de macros personnel d'Excel : les macros stockées dans ce modèle sont accessibles à tous les documents Word.

4. Cliquez sur le bouton Créer.

Visual Basic Editor s'ouvre sur la fenêtre Code de la macro MacroWord.

5. Complétez le code de la macro de la façon suivante :

```
Sub MacroWord()  
    MsgBox "Cette boîte de dialogue est affichée par la macro MacroWord", _  
        vbOKOnly + vbInformation, "Exécution d'une macro Word à partir d'un  
        ➡ programme Excel"  
End Sub
```

6. Enregistrez, puis fermez Word.

Retournez à Visual Basic Editor pour Excel et créez la procédure suivante :

```
1: Sub ExecuterMacroWord()  
2:     Dim MonWord As Object  
3:     Set MonWord = CreateObject("Word.Application")  
4:     MonWord.Visible = True  
5:     MonWord.Documents.Add  
6:     MonWord.Run "MacroWord"  
7:     Set MonWord = Nothing  
8: End Sub
```

Exécutez la procédure `ExecuterMacroWord`. La boîte de dialogue représentée à la Figure 6.15 s'affiche.

Ligne 2, la variable `MonWord` est créée et reçoit l'objet `Word.Application` à la ligne suivante. Ligne 4, la propriété `Visible` de l'objet `MonWord` est définie à `True` afin d'afficher Word à l'écran. Un document est ensuite créé. Ligne 6, on applique la méthode `Run` à l'objet `MonWord` afin d'exécuter la macro `MacroWord`. La boîte de dialogue représentée à la Figure 6.15 s'affiche alors. Ligne 7, la variable objet est libérée et le programme prend fin.

Figure 6.15

Une macro Excel peut contrôler l'exécution de macros dans d'autres applications hôte.



Le programme complet présenté au Chapitre 17 fournit un bon exemple de traitement interapplications, puisqu'il propose notamment d'éditer des documents Word complexes à partir de données traitées dans le programme Excel.



7

Contrôler les programmes VBA

Au sommaire de ce chapitre

- Répéter une série d'instructions : les boucles
- Utiliser des instructions conditionnelles
- Définir l'instruction suivante avec *GoTo*
- Interagir avec l'utilisateur *via* des boîtes de dialogue
- Utiliser les opérateurs logiques
- Trier des données

Visual Basic intègre des instructions permettant d'orienter le comportement d'une macro. Ces instructions sont appelées des *structures de contrôle* – on parle du flux de contrôle d'un programme. La connaissance et la maîtrise de ces structures constituent un préalable indispensable à la création de programmes VBA souples et puissants, se comportant différemment selon l'état du document et de l'application au cours de son exécution, ou des informations fournies par l'utilisateur.

Ce chapitre aborde une à une les structures de contrôle de Visual Basic. Leur combinaison permettra de gagner un temps précieux dans vos tâches les plus communes comme les plus complexes. L'instruction GoTo et les fonctions MsgBox et InputBox, ainsi que la collection Dialogs sont aussi traitées dans ce chapitre. Il ne s'agit pas de structures de contrôle, mais elles permettront aussi de contrôler le comportement des programmes VBA, et d'interagir avec l'utilisateur.

Répéter une série d'instructions : les boucles

On entend par *instructions en boucle*, des instructions se répétant en série. Des instructions en boucles peuvent se répéter un nombre de fois déterminé dans le code ou un nombre de fois indéterminé, en fonction du contexte au moment de l'exécution du programme.

- **Do...Loop** et **While...Wend**. Permettent de généraliser une série d'instructions particulières à l'ensemble d'un document ; dans ce cas, ce sont l'état du document et l'état de l'application qui déterminent le nombre de boucles réalisées.
- **For...Next**. Permet de répéter sur un document une série d'instructions un nombre de fois déterminé par l'utilisateur.
- **For Each...Next**. Permet d'exécuter une série d'instructions sur tous les objets d'une collection.

La boucle *While...Wend*

La structure de contrôle While...Wend permet de répéter une série d'instructions tant qu'une condition spécifiée est remplie. C'est l'une des structures les plus utilisées pour automatiser les tâches répétitives. Elle permet de répéter un traitement sur une chaîne, un format, un objet, etc., déterminé dans un document.

La syntaxe de la structure While...Wend est la suivante :

```
While Condition
    Série d'instructions
Wend
```

où *Condition* est une expression comparant deux valeurs à l'aide d'un opérateur relationnel. Lorsque la condition spécifiée après *While* est réalisée, le programme exécute la *Série d'instructions*, placée entre *While* et *Wend*. Lorsque l'instruction *Wend* est atteinte, le programme retourne à l'instruction *While* et interroge à nouveau la condition. Si elle est réalisée, la *Série d'instructions* s'exécute à nouveau, etc. Dans le cas contraire, les instructions placées entre *While* et *Wend* sont ignorées, et l'exécution du programme se poursuit avec l'instruction située immédiatement après *Wend*.

Pour poser une condition, on conjugue généralement une expression avec un *opérateur relationnel*, ou *opérateur de comparaison*, et une valeur. L'opérateur relationnel permet d'établir un rapport entre la valeur renvoyée par l'expression et la valeur qui lui est associée. Si ce rapport est vérifié, la condition est respectée.

Le Tableau 7.1 présente les opérateurs relationnels de Visual Basic.

Tableau 7.1 : Les opérateurs relationnels de Visual Basic

<i>Opérateur relationnel</i>	<i>Signification</i>
=	Égal à
>	Supérieur à
<	Inférieur à
<>	Différent de
>=	Supérieur ou égal à
<=	Inférieur ou égal à
Like	Identique à (pour comparer des chaînes de caractères)
Is	Égal à (pour comparer des variables objet)

Conseil

Deux chaînes de caractères peuvent être comparées à l'aide des opérateurs relationnels =, <, >, etc. La comparaison s'effectue alors entre les codes ANSI attachés aux caractères comparés. Si vous devez effectuer des comparaisons précises, préférez l'opérateur Like. Celui-ci permet en effet de prendre ou non en compte la casse et permet l'utilisation de caractères génériques. Consultez l'aide en ligne pour plus de précisions.

La technique d'enregistrement d'instructions en boucle la plus courante consiste à exécuter la *série d'instructions* après avoir activé l'Enregistreur de macro, puis à ouvrir la fenêtre Code de la macro et à y insérer la structure *While...Wend*.

Nous utiliserons une structure While...Wend pour automatiser la saisie d'informations dans une feuille de calcul. Considérez le classeur Représentants par départements représenté à la Figure 7.1. La feuille de calcul active (libellée Représentants) recense les départements affectés à chaque représentant de la société. Les noms des représentants apparaissent dans les cellules de la ligne 3. Pour chaque cellule contenant le nom d'un représentant, nous avons inséré en commentaires les initiales du représentant. Sous le nom du représentant se trouvent les numéros des départements dont il a la charge.

Figure 7.1

La répartition des représentants par départements.

	A	B	C	D	E	F	G	H
1	Répartition des représentants par départements							
2								
3	Valérie Marie	Hervé Dubéuf	Hélène Legrand	Ludovic Bidault	Noël CAPLIER			Joël Martin
4	75	10	09	18	01			45
5	92	21	12	22	03	05	08	77
6	95	25	16	28	07	06	14	94
7		39	17	29	15	11	27	78
8		51	19	35	26	13	59	91
9		52	23	36	38	30	60	
10		54	24	37	42	34	62	
11		55	31	41	43	48	76	
12		57	32	44	63	66	80	
13		58	33	49	69	83	93	
14		67	40	50	73	84		
15		68	46	53	74	98		
16		70	47	56				
17		71	64	61				

Le classeur Représentants par clients illustré à la Figure 7.2 contient la liste des clients de la société (colonne A). La colonne B nous renseigne sur la ville du client, et la colonne D sur son numéro de client. Les deux premiers chiffres de ce numéro correspondent au département d'origine du client. La colonne C contient les initiales du représentant en charge du client. Nous profiterons de ce que ces deux classeurs ont en commun le numéro du département pour automatiser la mise à jour de la colonne C.

Figure 7.2

Le classeur avec la répartition des représentants par clients avant mise à jour de la colonne C.

	A	B	C	D
1	Répartition des clients par représentant			
2				
3	Client	Ville	Représentant	N° Client
4	La Loterie	Joigny		8900002
5	Le Soldeur	Puteaux		9200003
6	Centre Soldes	Antibes		0600006
7	Prix Gros	Nantes		4400007
8	Le Grossiste	Saint Nazaire		4400015
9	La bonne affaire	La Rochelle		1700006
10	Le bon choix	Paris		7503009
11	Valable	Gannat		0300003
12	Facile	Marseille		1300046
13	C'est ici	Chalon		7100001
14	Prix écrasés	Colmar		6800014
15	Unique prix	Lyon		6900009

La macro suivante interroge les deux premiers chiffres du numéro de client. Elle recherche ensuite cette valeur dans le classeur *Representants* par département de façon à identifier le représentant en charge du client. Les initiales du client sont alors insérées dans la cellule située à gauche du numéro de client. La structure *While...Wend* permet de répéter cette procédure en boucle. Chaque fois que les initiales d'un représentant ont été insérées, la cellule Numéro de client suivante est activée. La procédure s'exécute **TANT QUE** la cellule sélectionnée contient une valeur.

```
1: Sub InsérerInitialesRepresentants()  
2:   Dim ClasseurRepresentants As Workbook  
3:   Dim NumDepartement As String  
4:   Dim Colonne As Variant  
5:   Dim Initiales  
6:   Set ClasseurRepresentants = _  
     GetObject("C:\Documents and settings\Administrateur\Bureau\Representants  
       par departements.xlsx")  
  
7:   Range("D4").Select  
8:   While ActiveCell.Value <>  
9:     NumDepartement = Left(ActiveCell.Value, 2)  
10:    Colonne = ClasseurRepresentants.Sheets(1).Range("A4:I50").  
        Find(What:=NumDepartement, LookIn:=xlFormulas, LookAt:=xlWhole).Address  
11:    Colonne = Range(Colonne).Column  
12:    Colonne = CInt(Colonne)  
13:    Initiales = ClasseurRepresentants.Sheets(1).Cells(3, Colonne).Comment.  
        Text  
14:    ActiveCell.Offset(0, -1).Range("A1").Select  
15:    ActiveCell.FormulaR1C1 = Initiales  
16:    ActiveCell.Offset(1, 1).Range("A1").Select  
17:   Wend  
18:   Set ClasseurRepresentants = Nothing  
19:   Workbooks("Representants par departements.xlsx").Close  
20: End Sub
```

Lignes 2 à 6, les variables qui seront exploitées par le programme sont déclarées, et à la variable objet *ClasseurReprésentant* est affecté le classeur *Representants* par départements.xlsx, situé sur le Bureau. Ligne 7, la cellule D4 est sélectionnée.

La boucle *While...Wend* des lignes 8 à 17 s'exécute tant que la cellule sélectionnée contient des informations. Ligne 9, la fonction *Left* affecte à la variable *NumDepartement* les deux caractères de gauche (correspondant au numéro de département) de la valeur de la cellule active. Cette valeur est ensuite recherchée dans le classeur des représentants par départements (ligne 10). L'objet *Range* renvoyé par la méthode *Find* est affecté à la variable *Colonne* – notez que la variable *Colonne* a été déclarée de type *Variant* de sorte qu'elle

puisse recevoir des valeurs de différents types. Ligne 11, Colonne reçoit la valeur correspondant au numéro de la colonne de la cellule trouvée. Ligne 12, la fonction CInt convertit la valeur de Colonne en valeur de type Integer. La variable Colonne peut ainsi être utilisée comme argument de la propriété Cells. Ligne 13, la variable Initiales reçoit pour valeur les initiales du représentant en charge du département. On lui affecte pour cela le texte de commentaires de la cellule située dans la même colonne que l'objet Range renvoyé par la fonction Find, mais sur la ligne 3 – la ligne des noms de représentants.

Lignes 14 et 16, un déplacement par référence relative aux cellules est effectué. Tout d'abord, la cellule située à gauche de la cellule active est sélectionnée et reçoit la valeur de la variable Initiales (ligne 15). Un déplacement d'une cellule vers la droite puis d'une cellule vers le bas est ensuite effectué. La cellule active est alors la cellule suivante de la colonne D. Le mot clé Wend renvoie l'exécution du programme à l'instruction While correspondante. Celle-ci vérifie que la cellule active contient des données. Si tel est le cas, les instructions situées entre While et Wend sont exécutées, et les initiales du représentant en charge du client insérées.

Lorsque la condition While n'est plus vérifiée, les instructions situées entre While et Wend sont ignorées et le programme se termine avec les instructions des lignes 18 et 19. Les ressources système occupées par la variable objet ClasseurReprésentants sont libérées, et le classeur des représentants par départements est fermé.



Pour sécuriser définitivement cette macro, commencez par lui faire activer la feuille devant recevoir les informations. Elle devra logiquement être stockée dans ClasseurReprésentants.xlsx, puisqu'elle ne servira qu'à ce classeur. Ainsi, elle ne s'exécutera que si le classeur est ouvert.

Figure 7.3

La macro a complété les informations de la colonne C.

	A	B	C	D	E
1	Répartition des clients par représentant				
2					
3	Client	Ville	Représentant	N° Client	Solde
4	La Loterie	Joigny	HD	8900002	
5	Le Soldeur	Puteaux	VM	9200003	
6	Centre Soldes	Antibes	MF	0600006	
7	Prix Gros	Nantes	LB	4400007	
8	Le Grossiste	Saint Nazaire	LB	4400015	
9	La bonne affaire	La Rochelle	HL	1700006	
10	Le bon choix	Paris	VM	7503009	
11	Valable	Gannat	NC	0300003	
12	Facile	Marseille	MF	1300046	
13	C'est ici	Chalon	HD	7100001	
14	Prix écrasés	Colmar	HD	6800014	
15	Unique prix	Lyon	NC	6900009	
16	Aux bas prix	Frejus	MF	8300001	
17	Exceptionnel	Colmar	HD	6800014	
18	Venez tous	Chalon	HD	7100001	
19					

La boucle *Do...Loop*

La structure de contrôle *Do...Loop* est semblable à *While...Wend*, mais offre plus de souplesse car elle peut se décliner sur quatre modes différents :

- **Do While...Loop.** Tant que la condition est respectée, la boucle s'exécute.

```
Do While Condition
    Série d'instructions
Loop
```

- **Do Until...Loop.** Jusqu'à ce que la condition soit réalisée, la boucle s'exécute.

```
Do Until Condition
    Série d'instructions
Loop
```

- **Do...Loop While.** La boucle s'exécute, puis se répète, tant que la condition est respectée.

```
Do
    Série d'instructions
Loop While Condition
```

- **Do...Loop Until.** La boucle s'exécute, puis se répète, jusqu'à ce que la condition soit respectée.

```
Do
    Série d'instructions
Loop Until Condition
```

Le programme suivant utilise une boucle *Do While...Loop* pour supprimer les doublons dans un classeur Excel. Les Figures 7.4 et 7.5 présentent un classeur contenant des doublons avant et après passage de la macro. On estime, dans cette première version, qu'il existe un doublon lorsque deux cellules de la colonne A contiennent les mêmes données. Le programme commence par faire un tri des données. Le contenu de chaque cellule de la colonne A est ensuite comparé à celui de la cellule suivante. S'ils sont identiques, la ligne de la cellule courante est supprimée.

```
1: Sub SuppressionDoublons()
2:   Dim CelluleCourante As Range
3:   Dim CelluleSuivante As Range
4:   Set CelluleCourante = ActiveSheet.Range("A1")
5:
6:   'Tri des données sur la cellule A1
7:   ActiveSheet.Range("A1").Sort key1:=Range("A1"), _
8:   Order1:=xlAscending, Header:= xlGuess, OrderCustom:=1, _
9:   MatchCase:=False, Orientation:=xlTopToBottom
```

```

10: 'Boucle
11: Do While Not IsEmpty(CelluleCourante) = True
12:   Set CelluleSuivante = CelluleCourante.Offset(1, 0)
13:   If CelluleSuivante.Value = CelluleCourante.Value Then
14:     CelluleCourante.EntireRow.Delete
15:   End If
16:   Set CelluleCourante = CelluleSuivante
17: Loop
18: End Sub

```

Lignes 2 et 3 les variables objet de type Range `CelluleCourante` et `CelluleSuivante` sont déclarées. La variable `CelluleCourante` reçoit ensuite un objet Range correspondant à la cellule A1 de la feuille active. L'instruction des lignes 7 à 9 trie les données. On applique pour cela la méthode `Sort`. Les arguments `Key1` et `Order1` définissent respectivement le premier critère de tri et l'ordre de tri. `Header` indique s'il existe des lignes de tri et reçoit ici la constante `xlGuess` (Excel définit s'il y a ou non une ligne de titre et, dans l'affirmative, de quelle ligne il s'agit). `OrderCustom` reçoit la valeur 1 et le tri est donc "Normal". Enfin `MatchCase` et `Orientation` correspondent au respect de la casse lors du tri et à son orientation (ici de haut en bas).

Lignes 11 à 17, une boucle `Do While...Loop` est utilisée pour tester toutes les cellules. La cellule stockée dans `CelluleCourante` est testée, puis `CelluleCourante` reçoit la cellule stockée dans `CelluleSuivante`, soit la cellule située immédiatement en dessous. La boucle s'exécute tant que la cellule stockée dans `CelluleCourante` n'est pas vide [`Not IsEmpty(CelluleCourante) = True`].

Ligne 12, la propriété `Offset` est utilisée pour attribuer à `CelluleSuivante` la cellule située une ligne en dessous, sur la même colonne. Lignes 13 à 15, une instruction conditionnelle sert à supprimer la ligne de `CelluleCourante` (`CelluleCourante.EntireRow`) si la cellule contient les mêmes données que la cellule suivante. `CelluleCourante` reçoit ensuite la cellule stockée dans `CelluleSuivante` (ligne 16).

Ligne 17, l'instruction `Loop` renvoie le programme à l'instruction `While` correspondante. L'expression `While` est de nouveau évaluée et le corps de la boucle s'exécute si elle est vérifiée. Lorsque l'expression de la ligne 11 n'est plus vérifiée, le programme se poursuit avec l'instruction située immédiatement sous l'instruction `Loop`. En l'occurrence, il prend fin.

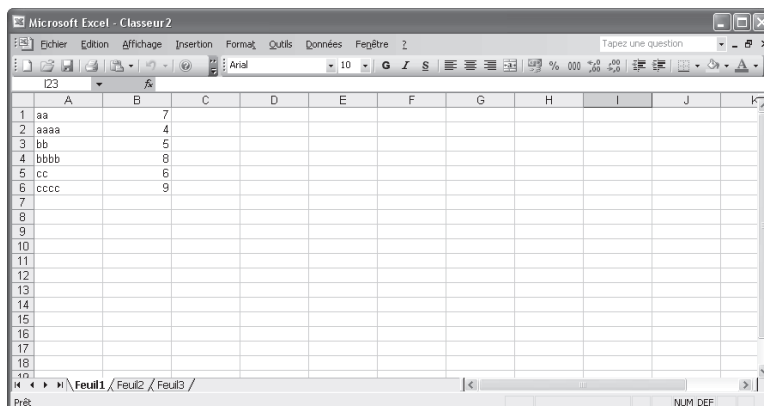
Figure 7.4

La feuille avant passage de la macro.

	A	B	C	D	E	F	G	H
1	aaaa	1						
2	bbbb	2						
3	cccc	3						
4	aaaa	4						
5	bb	5						
6	cc	6						
7	aa	7						
8	bbbb	8						
9	cccc	9						
10								
11								
12								
13								
14								

Figure 7.5

La macro a supprimé les doublons.



Le programme fonctionne correctement, mais ne prend en compte que le contenu des cellules de la colonne A pour déterminer les doublons. La procédure suivante supprime une ligne uniquement si les données sont également identiques dans les colonnes B, C et D. Les modifications apportées à la première version apparaissent en gras.

```

1: Sub SuppressionDoublons()
2:   Dim Cellulecourante As Range
3:   Dim Cellulesuivante As Range
4:   Set Cellulecourante = ActiveSheet.Range("A1")
5:
6:   'Tri des données sur la cellule A1
7:   ActiveSheet.Range("A1").Sort Key1:=Range("A1"), Order1:=xlAscending,
   Key2:=Range("B1"), _
8:   Order2:=xlAscending, Key3:=Range("C1"), Order3:=xlAscending,
   Header:=xlGuess, _
9:   OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
10:  'Boucle et test des cellules
11:  Do While Not IsEmpty(Cellulecourante) = True
12:    Set Cellulesuivante = Cellulecourante.Offset(1, 0)
13:    If Cellulesuivante.Value = Cellulecourante.Value Then
14:      If LignesIdentiques(Cellulecourante, Cellulesuivante) = True Then
15:        Cellulecourante.EntireRow.Delete
16:      End If
17:    End If
18:    Set Cellulecourante = Cellulesuivante
19:  Loop
20: End Sub
21:
22: Function LignesIdentiques(CellCourante As Range, CellSuivante As Range) As
Boolean

```



```

23:   If CellCourante.Offset(0, 1).Value <> CellSuivante.Offset(0, 1).Value
      Then
24:       LignesIdentiques = False
25:   ElseIf CellCourante.Offset(0, 2).Value <> CellSuivante.Offset(0, 2).Value
      Then
26:       LignesIdentiques = False
27:   ElseIf CellCourante.Offset(0, 3).Value <> CellSuivante.Offset(0, 3).Value
      Then
28:       LignesIdentiques = False
29:   Else
30:       LignesIdentiques = True
31:   End If
32: End Function

```

Ligne 15, l'instruction conditionnelle définissant si la ligne est supprimée appelle la fonction `LignesIdentiques` en lui passant les arguments `CelluleCourante` et `CelluleSuivante`. La fonction `LignesIdentiques` est déclarée comme recevant deux arguments de type `Range` et renvoyant une valeur de type `Boolean` (ligne 22).

Lignes 23 à 31, une structure conditionnelle sert à déterminer la valeur renvoyée par la fonction. Le contenu des cellules décalées d'une, de deux, puis de trois cellules à droite de `CelluleCourante` est successivement comparé au contenu des cellules décalées de la même façon par rapport à `CelluleSuivante` [`Offset(0, 1)`, `Offset(0, 2)` et `Offset(0,3)`]. Si ce contenu diffère, la valeur `False` est affectée à la fonction (lignes 24, 26 et 28). Dans le cas contraire, la fonction renvoie `True` (ligne 30). La procédure appelante reprend alors la main et l'instruction de la ligne 15 est exécutée si la fonction a renvoyé `True`. Dans le cas contraire, la condition n'est pas vérifiée et la cellule suivante est testée.



Pour interrompre une macro qui ne fonctionne pas correctement (qui exécute une boucle sans fin, par exemple), appuyez sur Ctrl+Pause, puis voyez le Chapitre 10.

La boucle **For...Next**

La structure de contrôle `For...Next` permet de répéter une série d'instructions un nombre de fois déterminé dans le code, en utilisant un compteur. Utilisez l'instruction `For...Next` selon la syntaxe suivante :

```

For compteur = x To y Step Pas
    série d'instructions
Next compteur

```

La macro exécute en boucle la série d'instructions spécifiée entre For et Next, en incrémentant la variable *compteur* de la valeur de *Pas* à chaque passage de la boucle. Si l'argument Step est omis, le compteur est incrémenté de 1. Tant que la valeur attachée à la variable *compteur* est inférieure à la valeur *y*, la boucle se répète ; lorsque la valeur numérique de *compteur* est supérieure à *y*, la boucle prend fin, et la procédure se poursuit avec les instructions situées derrière l'instruction Next.

La procédure suivante permet d'appliquer un ombrage de cellules à une ligne sur deux d'une feuille de calcul Excel, afin d'obtenir une mise en forme semblable à celle représentée à la Figure 7.6.

```
1: Sub FormaterClasseur()  
2:   Dim compteur As Integer  
3:   Dim MaLigne As Variant  
4:   Cells.Interior.ColorIndex = 2  
5:   MaLigne = Range("A1").End(xlDown).Address  
6:   MaLigne = Range(MaLigne).Row  
7:   If Not MaLigne / 2 = Int(MaLigne / 2) Then  
8:     MaLigne = MaLigne + 1  
9:   End If  
10:  For compteur = 2 To MaLigne Step 2  
11:    Range(compteur & ":" & compteur).Select  
12:    Selection.Interior.ColorIndex = 15  
13:  Next compteur  
14: End Sub
```

Deux variables sont tout d'abord déclarées. La propriété ColorIndex de l'objet Interior de tous les objets de la collection Cells (toutes les cellules de la feuille active) est ensuite définie à 2 (ligne 4) – ce qui revient à appliquer la couleur de fond blanche à l'ensemble des cellules de la feuille.

Les instructions des lignes 4 à 9 servent à déterminer jusqu'à quelle ligne le formatage doit s'effectuer. L'adresse de la dernière cellule non vide sous la cellule A1 est affectée à la variable MaLigne (ligne 5), qui reçoit ensuite pour valeur le numéro de ligne de cette cellule (ligne 6). Une instruction conditionnelle If...End If est utilisée pour vérifier que MaLigne est une valeur paire (lignes 7 à 9). Si tel n'est pas le cas (si MaLigne divisé par 2 n'est pas un nombre entier), MaLigne est incrémentée de 1.

La boucle For...Next peut maintenant être exécutée. Le compteur de la boucle commence à 2 et est incrémenté de 2 à chaque passage de la boucle, jusqu'à atteindre la valeur MaLigne (ligne 10). À chaque passage de la boucle, la ligne correspondant à la valeur de la variable compteur est sélectionnée (ligne 11) et l'ombrage de cellule correspondant à la valeur 15 de la propriété ColorIndex lui est appliqué (ligne 12).

Figure 7.6

Une mise en forme automatisée.

	A	B	C	D	E	F	G	H
1	Vendeur	Région	Ville	Chiffre d'A.				
2	Mary	Nord	Lille					
3	du Château	Ile de France	Paris					
4	Bi Do	Ouest	Quimper					
5	Des neury	Sud Ouest	Biarritz					
6	Gas Quai	Sud Est	Marseille					
7	Ussu Nez	Nord Ouest	Cherbourg					
8	Fernand	Centre	Saint Etienne					
9	Bagousse	Est	Châlon s/Saône					
10								
11								
12								

Le programme suivant constitue une nouvelle version de la procédure Suppression-Doublons, dans laquelle la fonction LignesIdentiques a été améliorée. La fonction utilise maintenant une boucle For...Next pour définir le déplacement (Offset) effectué lors des comparaisons. Par ailleurs, le nombre de cellules devant être comparées afin de définir si une ligne constitue un doublon peut ainsi être défini lors de l'appel de la fonction.

```

1: Sub SuppressionDoublons()
2:   Dim Cellulecourante As Range
3:   Dim Cellulesuivante As Range
4:   Set Cellulecourante = ActiveSheet.Range("A1")
5:
6:   'Tri des données sur la cellule A1
7:   ActiveSheet.Range("A1").Sort Key1:=Range("A1"), Order1:=xlAscending,
   Key2:=Range("B1"), _
8:   Order2:=xlAscending, Key3:=Range("C1"), Order3:=xlAscending,
   Header:=xlGuess, _
9:   OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
10:  'Boucle et test des cellules
11:  Do While Not IsEmpty(Cellulecourante) = True
12:    Set Cellulesuivante = Cellulecourante.Offset(1, 0)
13:    If Cellulesuivante.Value = Cellulecourante.Value Then
14:      If LignesIdentiques(Cellulecourante, Cellulesuivante, 3) = True
   Then
15:        Cellulecourante.EntireRow.Delete
16:      End If
17:    End If
18:    Set Cellulecourante = Cellulesuivante
19:  Loop
20: End Sub

```

```

21:
22: Function LignesIdentiques(CellCourante As Range, Cellsuivante As Range, Num
    As Byte) As Boolean
23:     LignesIdentiques = True
24:     Dim compteur As Byte
25:     'boucle et test du Num colonnes
26:     For compteur = 1 To Num
27:         If CellCourante.Offset(0, compteur).Value <> Cellsuivante.Offset
            (0, compteur).Value Then
28:             LignesIdentiques = False
29:             Exit For
30:         End If
31:     Next compteur
32: End Function

```

Ligne 14, la fonction `LignesIdentiques` est appelée et reçoit maintenant une valeur de type `Byte` pour l'argument `Num` (ici, 3).

La fonction `LignesIdentiques` contrôle ensuite *Num* cellules afin de définir si la ligne doit ou non être supprimée. Elle reçoit d'abord la valeur `True`. La boucle `For...Next` (lignes 26 à 31) s'exécute ensuite *Num* fois. Les cellules testées à chaque passage de la boucle correspondent à un déplacement de *Num* Cellules vers la droite. Si, lors d'un passage de la boucle, deux valeurs différentes sont décelées (ligne 27), la valeur `False` est affectée à `LignesIdentiques` et l'instruction `Exit For` entraîne la sortie de la boucle. Si les valeurs des cellules comparées sont toujours identiques, la boucle prend fin après *Num* passages, et la fonction garde la valeur `True`.

Boucles **For...Next** avec pas négatif

Le programme suivant supprime les lignes vides de la feuille active. Il utilise pour cela une structure `For...Next` avec un pas négatif de `-1`, de façon à parcourir l'ensemble des lignes de la feuille, de la dernière ligne employée jusqu'à la première.

```

1: Sub SupprLignesVides()
2:     Dim DerniereLigne As Long
3:     Dim Compteur As Long
4:     DerniereLigne = ActiveSheet.UsedRange.Row - 1 + _
        ActiveSheet.UsedRange.Rows.Count
5:     For Compteur = DerniereLigne To 1 Step -1
6:         If Application.WorksheetFunction.CountA(Rows(Compteur)) = 0 _
            Then Rows(Compteur).Delete
7:     Next Compteur
8: End Sub

```

Lignes 2 et 3, les variables sont déclarées. Ligne 4, on affecte à la variable `DerniereLigne` le numéro de la dernière ligne employée. On se sert pour cela de la propriété `UsedRange` qui renvoie la zone utilisée sur la feuille active, c'est-à-dire la zone rassemblant l'ensemble des cellules contenant des données sur la feuille. La propriété `Row` renvoie le numéro de la première ligne de cette zone. En retirant 1 à la valeur ainsi obtenue, on obtient le nombre de lignes vides précédant cette zone. L'expression `UsedRange.Rows.Count` renvoie le nombre de lignes de la zone. En additionnant ces deux valeurs, nous obtenons le numéro de la dernière ligne de la zone utilisée.

Une boucle `For...Next` est ensuite employée pour parcourir les lignes à vérifier. La boucle utilise un pas négatif (`Step -1`) de façon à effectuer le parcours de la dernière ligne à la première. Ligne 6, on vérifie si la ligne testée est vide. On utilise pour cela la fonction Excel `CountA` qui renvoie le nombre de cellules de la zone interrogée (ici la ligne entière, `Rows(Compteur)`) contenant des données. Si la ligne est vide (`CountA` renvoie 0), elle est supprimée. Le programme passe ensuite à la valeur suivante, en décrémentant notre compteur de 1.

L'utilisation d'un pas négatif nous assure ici que le programme teste toutes les lignes. En effet, si nous avions employé un pas positif et parcouru la zone utilisée de la première ligne à la dernière, la suppression d'une ligne aurait entraîné le décalage de la ligne suivante d'une ligne vers le haut, et elle n'aurait donc pas été traitée lors du passage suivant de la boucle.

Boucles *For...Next* imbriquées

Vous pouvez aussi utiliser des instructions `For...Next` imbriquées. Veillez simplement à donner des noms différents à chacune des variables compteur.

La procédure suivante utilise une structure `For...Next` pour stocker les valeurs du tableau présenté à la Figure 7.7 dans une variable de matrice multidimensionnelle.

Figure 7.7

Pour stocker les valeurs d'une feuille Excel dans une variable de matrice, utilisez des boucles `For...Next` imbriquées.

	A	B	C	D	E	F
1		Livres	Vidéo	Hi-Fi	Autres	
2	Janvier	58 963,00	45 225,00	85 485,00	45 225,00	
3	Février	45 895,00	32 568,00	79 658,00	32 568,00	
4	Mars	69 785,00	46 895,00	25 689,00	46 895,00	
5	Avril	45 214,00	54 897,00	49 652,00	54 897,00	
6	Mai	45 258,00	32 568,00	36 550,00	32 568,00	
7	Juin	38 652,00	97 632,00	56 320,00	97 632,00	
8	Juillet	32 510,00	45 863,00	45 520,00	45 863,00	
9	Août	28 952,00	32 568,00	47 965,00	32 568,00	
10	Septembre	45 693,00	94 625,00	29 865,00	94 625,00	
11	Octobre	48 956,00	31 582,00	16 495,00	31 582,00	
12	Novembre	65 920,00	21 458,00	75 632,00	21 458,00	
13	Décembre	95 120,00	12 589,00	12 589,00	12 589,00	
14						

```
Dim MonTableau() As Single

1: Sub BouclesForNextImbriquées()
2:   Dim DerniereLigne As Byte
3:   DerniereLigne = Range("A3").End(xlDown).Row
4:   Dim NbreDeLignes As Byte
5:   NbreDeLignes = DerniereLigne - 2
6:   ReDim MonTableau(NbreDeLignes, 4)
7:   Call AffecterValeursTableau(NbreDeLignes)
8: End Sub

9: Sub AffecterValeursTableau(DerniereLigneTableau)
10:  Dim CompteurLignes As Byte
11:  Dim CompteurColonnes As Byte
12:  For CompteurLignes = 1 To DerniereLigneTableau
13:    For CompteurColonnes = 1 To 4
14:      MonTableau(CompteurLignes, CompteurColonnes) = _
        Cells(CompteurLignes + 2, CompteurColonnes + 1)
15:    Next CompteurColonnes
16:  Next CompteurLignes
17: End Sub
```

La variable de matrice `MonTableau()` est déclarée dans la section Déclarations du module, afin d'être accessible à toutes les procédures du module. La procédure `BouclesFor-NextImbriquées` la redimensionne de sorte qu'elle accueille l'ensemble des données de la feuille Excel active. Pour un descriptif des instructions de cette procédure, reportez-vous à la section "Variables de matrice dynamiques" du chapitre précédent. Elle appelle ensuite la procédure `AffecterValeursTableau` en lui passant la valeur de la variable `NbreDeLignes`.

La procédure `AffecterValeursTableau` commence par créer deux variables numériques de type `Byte` qui serviront de compteur à chacune des boucles `For...Next` (lignes 10 et 11). La première boucle `For...Next` (lignes 12 à 16) utilise le compteur `CompteurLignes` pour répéter les instructions qui la composent un nombre de fois égal au nombre de lignes contenant des données à stocker. La boucle `For...Next` imbriquée (lignes 13 à 15) utilise le compteur `CompteurColonnes` pour répéter les instructions qui la composent une fois par colonne contenant des données à stocker. L'instruction qui la compose (ligne 14) affecte une valeur à un des espaces de stockage de la variable de matrice `MonTableau`. Les valeurs de ligne de la variable et de ligne de la feuille Excel sont liées à la valeur du compteur de la boucle `For...Next` principale, tandis que les valeurs de colonne de la variable et de colonne de la feuille Excel dépendent de la valeur du compteur de la boucle `For...Next` imbriquée. La variable reçoit les valeurs du tableau selon l'ordre suivant :

1. CompteurLigne = 1. La boucle For...Next imbriquée s'exécute quatre fois, et affecte les valeurs des cellules B3 à E3 à MonTableau(1, 1), MonTableau(1, 2), MonTableau(1, 3) et MonTableau(1, 4).
2. CompteurLigne = 2. La boucle For...Next imbriquée s'exécute quatre fois et affecte les valeurs des cellules B4 à E4 à MonTableau(2, 1), MonTableau(2, 2), MonTableau(2, 3) et MonTableau(2, 4).

[Etc.]

La boucle *For Each...Next*

Cette structure de contrôle permet de généraliser un traitement à l'ensemble des objets d'une collection et s'utilise selon cette syntaxe :

```
For Each élément In Collection
    Instructions
Next élément
```

où *élément* est une variable de type Object ou Variant, utilisée pour représenter chaque objet de la collection dans les *Instructions*, et *Collection* le nom de la collection d'objets. Les *Instructions* sont exécutées une fois pour chaque objet de la collection. La procédure suivante utilise une structure For Each...Next pour appliquer une couleur de police rouge (ColorIndex = 3) à tous les objets Cells de l'objet Selection (toutes les cellules de la sélection en cours) dont la valeur est supérieure à 1 000.

```
Sub FortesValeursEnRouge()
    Dim cellule As Range
    For Each cellule In Selection.Cells
        If cellule.Value > 1000 Then
            cellule.Font.ColorIndex = 3
        End If
    Next cellule
End Sub
```

La procédure suivante enregistre tous les classeurs Excel ouverts – à l'exception du classeur PERSONAL.XLSB – au format Excel 4.0 et les ferme :

```
1: Sub EnregistrerFormatExcel4EtFermer()
2:   Dim Classeur As Workbook
3:   Dim position As Byte
4:   Dim NomClasseur As String
5:   For Each Classeur In Workbooks
6:     If Not Classeur.Name = "PERSONAL.XLSB" Then
7:       NomClasseur = Classeur.FullName
```

```
8:         position = InStr(NomClasseur, ".xlsx")
9:         NomClasseur = Left(NomClasseur, position - 1) & ".xlw"
10:        Classeur.SaveAs FileName:=NomClasseur, _
           FileFormat:=xlExcel4Workbook
11:        Classeur.Close
12:    End If
13: Next Classeur
14: End Sub
```

Les lignes 2 à 4 déclarent les variables nécessaires au programme. La condition de la ligne 6 vérifie que le classeur Excel correspondant à l'objet `Classeur` n'est pas le classeur de macros personnel `PERSONAL.XLSB`. On utilise pour cela l'opérateur logique `Not` (présenté à la fin de ce chapitre), et la propriété `Name` qui, attachée à un objet `Workbook`, renvoie le nom de fichier de ce dernier (sans le chemin).

Ligne 7, la variable `NomClasseur` se voit affecter le nom complet du classeur – le chemin suivi du nom de fichier – renvoyé par la propriété `FullName`. Les fonctions `InStr` et `Left` sont utilisées pour substituer l'extension `.xlw` à l'extension `.xlsx`, afin de définir des noms d'enregistrement corrects pour les fichiers. La fonction `InStr` sert à comparer deux chaînes de caractères. Ici, elle renvoie une valeur numérique représentant la position de la chaîne `".xlsx"` dans la chaîne stockée dans la variable `NomClasseur`. La fonction `Left` sert à renvoyer un nombre déterminé de caractères situés à gauche d'une chaîne. Ici, elle renvoie le nombre de caractères égal à $(\text{position} - 1)$ dans la chaîne `NomClasseur`. On obtient ainsi le nom du document sans l'extension `".xlsx"`. Il suffit alors de concaténer la valeur de `NomClasseur` et la chaîne `".xlw"` pour obtenir un nom de fichier possédant l'extension `".xlw"`. La méthode `SaveAs` est ensuite appliquée à l'objet `Classeur`, en affectant à l'argument nommé `FileName` le nom ainsi défini, et à l'argument nommé `FileFormat`, la constante Excel `xlExcel4Workbook`. Enfin, la méthode `Close` ferme le document ainsi enregistré.

Info

Le traitement des chaînes de caractères est un sujet incontournable de la programmation. Vous serez inévitablement amené à manipuler des chaînes de caractères (composées de lettres comme de chiffres) afin d'en extraire les données voulues ou de les modifier.

Les fonctions de traitement des chaînes de caractères sont présentées au Chapitre 11.

Boucles **For Each...Next** imbriquées

À l'instar des boucles `For...Next` et, plus largement, de l'ensemble des structures de contrôle, vous pouvez imbriquer des structures de contrôle `For Each...Next`. L'exemple suivant extrait l'ensemble des formules du classeur inscrit et les écrit dans un document

Word qui est ensuite imprimé. Nous utilisons pour cela deux structures `For Each...Next`. La première parcourt la collection des feuilles de travail (`ActiveWorkbook.Worksheets`) du classeur, tandis que la seconde y est imbriquée et parcourt la collection des cellules de la zone courante définie à partir de la cellule A1 (`MaFeuille.Cells(1, 1).CurrentRegion.Cells`). Lorsque la boucle imbriquée a fini de traiter les cellules de la zone courante de la feuille de travail en cours de traitement, la première structure `For Each...Next` reprend la main et traite donc l'objet `Worksheet` – la feuille de travail – suivant de la collection.

```
1: Public Sub ExtraireMaFormulesWord()
2:   Dim MaFormule As String
3:   Dim MaCellule As Range
4:   Dim MaFeuille As Worksheet
5:   Dim MonWord As Object
6:   On Error Resume Next
7:   Set MonWord = GetObject(, "Word.Application")
8:   If Err.Number <> 0 Then
9:     Set MonWord = CreateObject("Word.Application")
10:    Err.Clear
11:  End If
12:  MonWord.Visible = True
13:  MonWord.Documents.Add

14:  For Each MaFeuille In ActiveWorkbook.Worksheets
15:    With MonWord.Selection
16:      .Font.Name = "Arial"
17:      .Font.Bold = True
18:      .Font.Size = "13"
19:      .TypeText "Formules de la feuille : " & MaFeuille.Name & Chr(13)
20:      .Font.Size = 14
21:      .Font.Bold = False
22:    End With
23:    For Each MaCellule In MaFeuille.Cells(1, 1).CurrentRegion.Cells
24:      If MaCellule.HasFormula = True Then
25:        MaFormule = "{" & MaCellule.Formula & "}"
26:        MonWord.Selection.TypeText "Cellule " & _
27:          MaCellule.Address(False, False, xlA1) & _
28:          " : " & MaFormule & Chr(13)
29:      End If
30:    Next MaCellule
31:  Next MaFeuille

32: End Sub
```

Lignes 2 à 5, les variables sont déclarées. Lignes 6 à 13, la variable MonWord reçoit l'objet Word.Application. Notez que la méthode GetObject est utilisée avec un gestionnaire d'erreur afin de capturer l'erreur générée si Word n'est pas ouvert. Auquel cas, la méthode CreateObject crée une nouvelle instance de l'application Word (ligne 9), et l'objet Err qui reçoit l'erreur est réinitialisé (ligne 10). Lignes 12 et 13, l'application Word s'affiche et un nouveau document est créé.

Lignes 14 à 31, la première boucle For Each...Next parcourt la collection des feuilles du classeur. Pour chaque feuille parcourue, le document Word reçoit un texte formaté (lignes 15 à 22). Nous utilisons pour cela une structure With...End With qui définit la police employée (Arial, corps 13) avant d'insérer le texte "Formules de la feuille : " suivi du nom de la feuille. La taille de la police est ensuite redéfinie à 11.

La seconde structure For Each...Next (lignes 23 à 30) prend alors la main et traite chacune des cellules de la collection MaFeuille.Cells(1, 1).CurrentRegion.Cells. Notez que l'on utilise ici la propriété CurrentRegion pour définir la zone courante à partir de la cellule A1 (Cells(1, 1)) de la feuille. À chaque occurrence de notre boucle, on vérifie si la cellule contient une formule (ligne 24), et si tel est le cas, celle-ci est insérée dans le document Word, précédée de l'adresse de la cellule concernée.

Lorsque toutes les cellules ont été traitées, la boucle imbriquée prend fin et l'instruction de la ligne 31 appelle le passage suivant de la structure For Each...Next principale.

Figure 7.8
*Toutes les formules
du classeur ont été extraites.*

```
Formules de la feuille : Feuil1
Cellule B11 : {=SUM(B3:B10)}
Cellule C11 : {=SUM(C3:C10)}
Cellule D11 : {=SUM(D3:D10)}
Cellule B12 : {=AVERAGE(B3:B10)}
Cellule C12 : {=AVERAGE(C3:C10)}
Cellule D12 : {=AVERAGE(D3:D9)}
Cellule B13 : {=PRODUCT(B12:30)}
Cellule C13 : {=PRODUCT(C12:30)}
Cellule D13 : {=PRODUCT(D12:30)}
Cellule B15 : {=PRODUCT(B13,1:33)}
Cellule C15 : {=PRODUCT(C13,1:33)}
Cellule D15 : {=PRODUCT(D13,1:33)}
Cellule B16 : {=PRODUCT(B15,1:2)}
Cellule C16 : {=PRODUCT(C15,1:2)}
Cellule D16 : {=PRODUCT(D15,1:2)}
Cellule B17 : {=PRODUCT(B16,1:1)}
Cellule C17 : {=PRODUCT(C16,1:1)}
Cellule D17 : {=PRODUCT(D16,1:1)}
Formules de la feuille : Feuil2
Cellule B11 : {=SUM(B3:B10)}
Cellule C11 : {=SUM(C3:C10)}
Cellule D11 : {=SUM(D3:D10)}
Cellule B12 : {=AVERAGE(B3:B10)}
Cellule C12 : {=AVERAGE(C3:C10)}
Cellule D12 : {=AVERAGE(D3:D9)}
Cellule B13 : {=PRODUCT(B12:30)}
Cellule C13 : {=PRODUCT(C12:30)}
Cellule D13 : {=PRODUCT(D12:30)}
Cellule B15 : {=PRODUCT(B13,1:33)}
Cellule C15 : {=PRODUCT(C13,1:33)}
Cellule D15 : {=PRODUCT(D13,1:33)}
Formules de la feuille : Feuil3
```

Utiliser des instructions conditionnelles

Si les boucles permettent de réaliser des macros puissantes, l'usage des conditions leur assurera souplesse et sûreté. Les instructions conditionnelles permettent entre autres choses de :

- s'assurer que l'environnement de l'application et l'état du document sont compatibles avec l'exécution du programme ;
- modifier le comportement de la macro en fonction de l'état du document et de l'application à un moment précis ;
- échanger des informations avec l'utilisateur lors de l'exécution du programme (combinées avec la fonction MsgBox, par exemple).

La structure de contrôle *If...Then...Else*

Au même titre que *While...Wend*, *If...Then Else* est une instruction conditionnelle. Cependant, cette structure est plus souple et son utilisation comme structure conditionnelle plus répandue que celle de *While...Wend*, essentiellement utilisée pour réaliser des boucles. La structure *If...Then...Else* permet en effet de spécifier différentes options d'exécution dans une procédure, en fonction de l'état de l'application ou/et du document, ou de conjuguer les conditions dans des boucles imbriquées.

Dans sa forme minimale, l'instruction conditionnelle *If* se présente ainsi :

```
If Condition Then
    Série d'instructions
End If
```

Lorsque la *Condition* spécifiée est remplie, la *Série d'instructions* est exécutée ; sinon, ces instructions sont ignorées et la procédure se poursuit avec l'instruction située immédiatement après l'instruction *End If*.

La macro *Auto_Open* suivante utilise une structure *If...End If* pour contrôler l'affichage de la boîte de dialogue présentée à la Figure 7.9. Elle utilise pour cela la fonction *Date* qui renvoie la date du jour.



Une macro Auto_Open s'exécute automatiquement à l'ouverture du classeur Excel dans lequel elle est stockée.

```
Sub Auto_Open()
    If Date > "30/11/07" and Date < "08/12/07" Then
        MsgBox "Attention ! Remise des budgets " & _
```

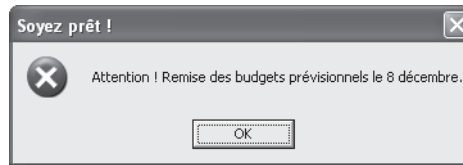
```

        "prévisionnels le 8 décembre.", _
        vbOKOnly + vbCritical, "Soyez prêt !"
    End If
End Sub

```

Figure 7.9

Ce message s'affiche à chaque ouverture du classeur effectuée entre le 1^{er} et le 7 décembre.



La valeur attachée à l'opérateur relationnel dans une condition varie selon l'objet de la comparaison ; il peut s'agir d'une chaîne de caractères si l'objet de la condition est la valeur renvoyée par une variable de type String ou une expression renvoyant elle-même une chaîne de caractères. Il s'agira d'une valeur numérique si l'objet de la condition est une variable numérique ou une expression renvoyant une valeur numérique, ou encore d'une valeur de type Boolean si l'objet de la condition est une variable de type Boolean ou une expression renvoyant une valeur de ce type.

Dans l'exemple suivant, l'instruction If permet de s'assurer que les conditions nécessaires au bon fonctionnement du programme sont réalisées (en l'occurrence que deux fenêtres de document sont ouvertes). Si ce n'est pas le cas, un message s'affiche à l'attention de l'utilisateur et l'instruction Exit Sub entraîne la sortie de la procédure.

```

Sub VérifierConditions
    If Workbooks.Count <> 2 Then
        MsgBox "La macro ne peut être exécutée. " & _
            "Deux classeurs doivent être ouverts."
        Exit Sub
    End if
    'Instructions exécutées si deux classeurs sont ouverts
End Sub

```

Une structure If...Then...Else autorise un nombre de conditions indéterminé. Vous pouvez ainsi envisager les différents cas possibles dans une situation particulière et indiquer à la procédure les instructions à exécuter dans chacun de ces cas.

L'instruction répond alors à la syntaxe suivante :

```

If condition1 Then
    Série d'instructions 1
ElseIf condition2 Then
    Série d'instructions 2
ElseIf condition3 Then

```

```
Série d'instructions 3
...
Else
    Série d'instructions n
End If
```

Contrairement à ElseIf, l'instruction Else ne pose aucune condition : elle apparaît en dernière position et les instructions qui lui sont attachées sont automatiquement exécutées si aucune des conditions posées auparavant n'a été réalisée. Par contre, si l'une des conditions posées par une instruction If ou ElseIf est réalisée, la macro exécute les instructions qui lui sont attachées, puis ignore les autres conditions posées et se poursuit avec les instructions situées après End If.

Les instructions ElseIf comme Else sont facultatives. Une instruction conditionnelle peut être composée d'une ou de plusieurs instructions ElseIf et ne pas présenter d'instruction Else, et inversement. La fonction suivante détermine la valeur d'une remise sur un achat, et insère cette valeur ainsi que le prix après remise dans la feuille de calcul.

```
1: Sub CalculRemiseEtPrixDefinitif()
2:   Dim PrixAvantRemise As Single, PrixDefinitif As Single
3:   PrixAvantRemise = ActiveSheet.Range("C11")
4:   PrixDefinitif = PrixAvecRemise(PrixAvantRemise)
5:   ActiveSheet.Range("C13").Value = PrixDefinitif
6: End Sub

7: Function PrixAvecRemise(ValeurAchat)
8:   Dim PourcentageRemise As Single
9:   If ValeurAchat <= 1000 Then
10:    PourcentageRemise = 0
11:   ElseIf ValeurAchat > 1000 And ValeurAchat <= 2000 Then
12:    PourcentageRemise = 0.1
13:   ElseIf ValeurAchat > 2000 And ValeurAchat <= 5000 Then
14:    PourcentageRemise = 0.2
15:   ElseIf ValeurAchat > 5000 And ValeurAchat < 10000 Then
16:    PourcentageRemise = 0.25
17:   Else
18:    PourcentageRemise = 0.3
19:   End If
20:   ActiveSheet.Range("C12").Value = PourcentageRemise
21:   PrixAvecRemise = ValeurAchat - (ValeurAchat * PourcentageRemise)
22: End Function
```

La procédure CalculRemiseEtPrixDefinitif déclare les variables PrixAvantRemise et PrixDefinitif de type Single. La variable PrixAvantRemise reçoit la valeur de la

cellule C11 de la feuille active. L'instruction de la ligne 4 appelle la fonction `PrixAvecRemise` en lui passant la valeur de `PrixAvantRemise`.

La structure conditionnelle `If...Then...Else` des lignes 9 à 19 affecte une valeur à la variable `PourcentageRemise`, fonction de la valeur de `ValeurAchat`. La valeur de la remise est insérée dans la cellule C12. Ligne 21, la fonction reçoit la valeur après remise, c'est-à-dire la valeur de `ValeurAchat` moins le prix de la remise (`ValeurAchat * PourcentageRemise`).

La procédure principale reprend ensuite la main. L'instruction de la ligne 5 affecte alors à la cellule C13 la valeur de `PrixDefinitif`. La procédure prend fin.



Une instruction conditionnelle peut aussi s'écrire sur une seule ligne, en utilisant deux points (:) comme séparateurs entre les instructions à exécuter si la condition est vérifiée. L'instruction `End If` est alors omise et la syntaxe est la suivante :

`If Condition Then Instruction1 : Instruction2 : ... : InstructionN`

Par exemple, l'instruction :

```
If Selection.Font.Italic() = True Then  
Selection.Font.Italic() = False  
End If
```

est aussi valide sous la forme :

```
If Selection.Font.Italic() = True Then Selection.Font.Italic() =  
False
```

Pour présenter plusieurs instructions sur une même ligne, utilisez les deux points (:) comme séparateur :

```
If Selection.Font.Italic() = True Then Selection.Font.Italic() =  
False : Selection.Font.Bold = True
```

Conditions imbriquées

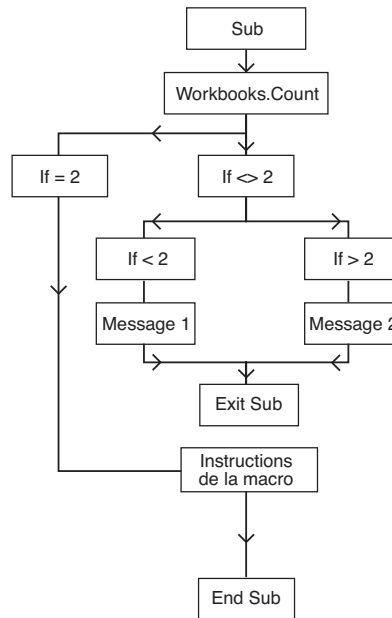
Vous pouvez définir des conditions à l'intérieur d'une condition initiale. On parle alors de *conditions imbriquées*. Les conditions imbriquées permettent de prendre en considération un grand nombre de possibilités lors de l'exécution du programme.

L'exemple suivant est composé d'une première instruction conditionnelle qui vérifie que deux fenêtres sont ouvertes avant de s'exécuter. Nous y avons *imbriqué* une instruction conditionnelle (en gras, dans le texte de la macro), afin que la boîte de dialogue affichée si la première condition est réalisée (`Workbooks.Count <> 2`) varie en fonction du nombre de fenêtres ouvertes.

L'organigramme de la Figure 7.10 présente la structure de cette macro.

Figure 7.10

Les conditions imbriquées assurent aux macros souplesse et précision.

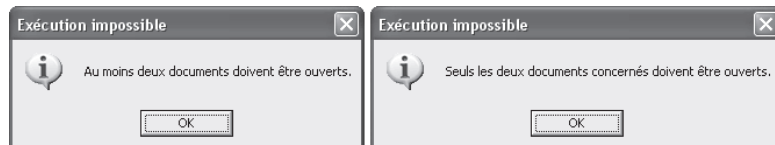


```

Sub ConditionsImbriquées()
    If Workbooks.Count <> 2 Then
        Dim Message As String
        If Workbooks.Count < 2 Then
            Message = "Au moins deux documents doivent être ouverts."
        Else
            Message = "Seuls les deux documents concernés doivent être ouverts."
        End If
        MsgBox Message, vbOKOnly + vbInformation, "Exécution impossible"
        Exit Sub
    End if
    Instructions de la macro
End Sub
  
```

Figure 7.11

L'instruction conditionnelle imbriquée détermine le message qui sera affiché.



La structure de contrôle *Select Case*

La structure de contrôle *Select Case* permet d'envisager différentes valeurs pour une même expression et de déterminer des instructions spécifiques pour chaque cas envisagé. Elle répond à la syntaxe suivante :

```
Select Case Expression
  Case valeur1
    Instructions
  Case valeur2
    Instructions
  ...
  Case valeurn
    Instructions
  Case Else
    Instructions
End Select
```

où *Expression* est une expression renvoyant une valeur dont le type peut varier. Lorsque la valeur renvoyée par *Expression* correspond à l'une des valeurs posées par les instructions *Case*, les *Instructions* correspondantes s'exécutent, et la procédure se poursuit avec l'instruction qui suit *End Select*. Si aucune des valeurs posées par les instructions *Case* ne correspond à la valeur renvoyée par *Expression*, les instructions attachées à *Case Else* sont exécutées. L'utilisation de *Case Else* est facultative.

Le programme suivant détermine la valeur de la variable *Reduction*, fonction de la valeur de la cellule D7 de la feuille 1 du classeur *commande.xlsx*. Une boîte de dialogue est ensuite affichée, afin d'informer l'utilisateur de la remise qui sera effectuée.

```
1: Sub AffichageReduction()
2:   Dim Reduction As Variant
3:   Dim LongueurChaîne As Byte
4:   Reduction = CalculerValeurReduction _
      (Workbooks("commande.xlsx").Sheets(1).Range("D7").Value)
5:   LongueurChaîne = Len(Reduction)
6:   If LongueurChaîne = 3 Then Reduction = Reduction & "0"
7:   MsgBox "La remise effectuée sera de " & Reduction & " %."
8: End Sub

9: Function CalculerValeurReduction(PrixCommande)
10:  Select Case PrixCommande
11:    Case 0 To 999.99
12:      CalculerValeurReduction = 0
13:    Case 1000 To 1999.99
```



```
14:      CalculerValeurReduction = 0.1
15:      Case 2000 To 2999.99
16:      CalculerValeurReduction = 0.25
17:      Case Else
18:      CalculerValeurReduction = 0.4
19:  End Select
20: End Function
```

La variable `Reduction` est tout d'abord déclarée de type `Variant`. Elle stockera en effet une valeur numérique, qui sera ensuite manipulée en tant que chaîne de caractères. L'instruction d'affectation de la ligne 4 appelle la fonction `CalculerValeurReduction`, en lui passant la valeur de la cellule D7 de la première feuille du classeur `commande.xlsx`.

La fonction `CalculerValeurReduction` utilise une structure `Select Case` pour renvoyer une valeur fonction de la valeur de l'argument `PrixCommande` (ici la valeur de la cellule D7). Le mot clé `To` est utilisé pour définir des plages de valeurs (0 à 999.99, 1000 à 1999.99, etc.). La procédure principale reprend ensuite la main.

Les instructions des lignes 5 et 6 servent à formater la chaîne stockée dans la variable `Reduction`. La fonction `Len` renvoie la longueur (le nombre de caractères) de la variable `Reduction`, qui est stockée dans la variable `LongueurChaîne`. Si la chaîne comprend trois caractères, un 0 est ajouté à la fin de la chaîne `Reduction` (0,1 et 0,4 deviennent respectivement 0,10 et 0,40). Enfin, l'instruction de la ligne 7 affiche une boîte de dialogue informant l'utilisateur de la valeur de la remise qui sera effectuée.



Dans les instructions Visual Basic, le point est utilisé comme séparateur décimal des valeurs numériques. Cependant, lorsque vous affichez sous forme de chaîne une valeur numérique – comme l'instruction de la ligne 7 de l'exemple précédent –, la virgule est utilisée comme séparateur de décimales.

Définir l'instruction suivante avec **GoTo**

L'instruction `GoTo` permet d'orienter le déroulement d'une procédure vers l'emplacement spécifié par l'utilisateur, et ce à tout moment de l'exécution. Cette instruction s'utilise avec une étiquette, c'est-à-dire une balise placée dans le texte. La syntaxe de `GoTo` est la suivante :

`GoTo Etiquette`

L'étiquette spécifiée après `GoTo` doit être placée en début d'une ligne indépendante, située avant l'instruction sur laquelle on veut brancher la procédure et être immédiatement suivie des deux points ":". Une instruction `GoTo` ne peut renvoyer qu'à une étiquette se trouvant

dans la même procédure que l'instruction elle-même. Les instructions GoTo rendent difficile la lecture du code. Préférez les structures de contrôle à l'instruction GoTo.

Interagir avec l'utilisateur *via* des boîtes de dialogue

L'affichage de boîtes de dialogue au cours de l'exécution d'un programme permet de renseigner l'utilisateur sur son déroulement, ou de lui demander des informations qui en modifieront le cours. Deux fonctions permettent l'affichage de boîtes de dialogue :

- **InputBox.** Entraîne l'affichage d'une boîte de dialogue présentant une zone de texte dans laquelle l'utilisateur est invité à entrer des informations.
- **MsgBox.** Permet d'afficher un message à l'attention de l'utilisateur dans une boîte de dialogue et de lui proposer de choisir entre différentes possibilités en cliquant sur l'un des boutons de commande affichés.

La fonction *InputBox*

La fonction VBA InputBox permet d'afficher une boîte de dialogue contenant une zone de texte légendée, afin d'inviter l'utilisateur à y saisir l'information attendue ; l'information saisie est renvoyée sous forme de chaîne de caractères et stockée dans une variable afin de pouvoir être ensuite exploitée par le programme.

La fonction InputBox s'utilise selon la syntaxe suivante :

```
InputBox(prompt, title, default)
```

prompt, *title* et *default* sont des arguments nommés de type String. *prompt* est le message affiché dans la boîte de dialogue afin de légender la zone de texte. L'argument *title* correspond au titre de la boîte de dialogue (affiché dans la barre de titre) et est facultatif. Si cet argument est omis, le nom de l'application apparaît dans la barre de titre. *default* est aussi facultatif et correspond au texte apparaissant par défaut dans la zone de texte lors de l'affichage de la boîte de dialogue.

La procédure suivante affiche la boîte de dialogue représentée à la Figure 7.12.

```
Sub UtilisationDeInputBox()  
    Dim DateVal As String  
    DateVal = InputBox("Date de validité :", "Nouveau membre", Date)  
End Sub
```

Figure 7.12

La fonction `InputBox` permet de stocker des informations entrées par l'utilisateur durant l'exécution du programme.



Pour stocker l'information fournie par l'utilisateur dans une variable, il suffit d'affecter une variable à la fonction, selon la syntaxe suivante :

```
Variable = InputBox(prompt, title, default)
```

où *Variable* est une variable de type `String` ou `Variant`.



Si l'affichage d'une réponse par défaut n'est pas indispensable, il peut se révéler fort pratique pour orienter l'utilisateur lorsque l'information demandée autorise plusieurs formats. Par exemple, si vous demandez à l'utilisateur d'entrer une date, et s'il est nécessaire, pour le bon déroulement de la macro, que celle-ci soit saisie sous le format jj/mm/aa. L'affichage d'une date hypothétique à l'aide de l'argument `default` permettra d'indiquer à l'utilisateur le format attendu.

Vous pouvez afficher des variables ou des caractères réservés (comme le guillemet ou la virgule) dans une boîte de dialogue. On utilise pour ceci :

- l'opérateur de concaténation `&` ;
- la fonction `Chr`, pour afficher un caractère réservé en spécifiant le code ASCII (entre 1 et 32).

La fonction `InputBox` se présente alors ainsi :

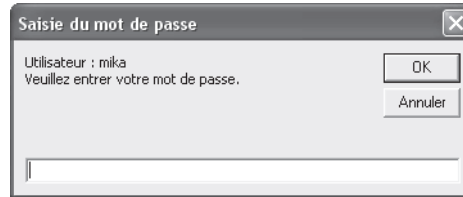
```
InputBox("Texte" & variable + Chr(CodeAscii) + "Texte", "Titre", "Entrée par défaut")
```

La procédure suivante affiche la boîte de dialogue présentée à la Figure 7.13 :

```
Sub OuvertureSession()
    Dim Utilisateur As String, MotDePasse As String
    Utilisateur = Application.UserName
    MotDePasse = InputBox("Utilisateur : " & Utilisateur & Chr(10) & _
        "Veuillez entrer votre mot de passe.", "Saisie du mot de passe")
    Call VerifierMotDePasse(Utilisateur, MotDePasse)
End Sub
```

Figure 7.13

Utilisez l'opérateur de concaténation & pour intégrer des variables ou des caractères réservés dans les chaînes de caractères.



Dans l'exemple suivant la fonction InputBox sert à demander à l'utilisateur d'indiquer une date d'échéance pour les opérations en cours. La procédure met ensuite en valeur les cellules sélectionnées dont la date est supérieure à la date indiquée.

```

1: Sub VerifierEcheances()
2:   'Vérifier qu'il existe une plage de cellules est sélectionnée
3:   Dim ZoneATester As String
4:   ZoneATester = ActiveWindow.RangeSelection.Address
5:   If ZoneATester = Null Then
6:     MsgBox "Sélectionnez la plage de cellules à tester.", _
           vbOKOnly & vbInformation
7:     Exit Sub
8:   End If

9:   'Demander à l'utilisateur la date d'échéance
10:  Dim DateEcheance As Variant
11:  DateEcheance = InputBox("Indiquez la date d'échéance.", _
    "Echéance des opérations en cours", Date + 30)
12:  DateEcheance = CDate(DateEcheance)

13:  'Tester toutes les cellules de la sélection
14:  Dim CellTest As Range
15:  For Each CellTest In Range(ZoneATester)
16:    If IsDate(CellTest) = True Then
17:      If CellTest.Value > DateEcheance Then
18:        CellTest.Interior.ColorIndex = 6
19:      End If
20:    End If
21:  Next
22: End Sub

```

La procédure commence par vérifier qu'une plage de cellules a été sélectionnée dans le classeur actif (lignes 2 à 8). Elle affecte pour cela l'adresse de la sélection en cours à la variable ZoneATester. Si aucune zone n'est sélectionnée (ZoneATester = Null), un message s'affiche à l'attention de l'utilisateur et l'instruction de la ligne 7 entraîne la sortie de la procédure.

La fonction InputBox sert ensuite à demander à l'utilisateur d'indiquer une date d'échéance (ligne 11). La fonction Date est utilisée pour déterminer la valeur par défaut de la zone de texte. Elle renvoie la date du jour à laquelle on ajoute 30 jours. La chaîne renvoyée par la fonction InputBox est stockée dans la variable DateEcheance de type Variant. Ligne 12, la fonction CDate est utilisée pour convertir la variable DateEcheance en une variable de type Date.



Si vous déclarez la variable de DateEcheance de type String, le programme fonctionnera correctement, mais l'instruction de conversion de type de données de la ligne 12 ne modifiera pas le type de la variable DateEcheance. L'instruction conditionnelle de la ligne 17 effectuera alors une comparaison entre les chaînes de caractères et non entre les dates. Le programme se déroulera correctement, mais produira des résultats erronés.

La procédure teste ensuite l'ensemble des cellules sélectionnées. Une variable objet de type Range est déclarée ligne 14. Lignes 15 à 22, une structure de contrôle For Each...Next est utilisée pour tester tous les objets Range de la collection d'objets contenue dans la sélection en cours.

L'instruction conditionnelle de la ligne 16 vérifie que la cellule traitée contient des données de type Date. Si ce n'est pas le cas, elle est ignorée, et la boucle se poursuit avec la cellule suivante. Si la cellule contient des données de type Date, la structure conditionnelle des lignes 17 à 19 est exécutée : si la valeur de la cellule est supérieure à la date stockée dans la variable DateEcheance, la couleur jaune (ColorIndex = 6) est appliquée à l'intérieur de la cellule. La Figure 7.14 représente une feuille de calcul après passage de la macro VerifierEcheances (l'utilisateur a indiqué le 01/04/98 pour date d'échéance).

Figure 7.14

L'ensemble des cellules dont la date est supérieure à la date indiquée par l'utilisateur est mis en évidence.

	A	B	C	D	E	F	G	H	I
1	26/03/98	10/03/98	13/03/98	18/03/98	23/03/98	08/05/98	17/04/98	22/04/98	27/04/98
2	10/04/98	27/03/98	02/04/98	06/04/98	08/04/98	27/03/98	09/03/98	13/03/98	18/03/98
3	26/03/98	10/03/98	13/03/98	18/03/98	23/03/98	27/03/98	10/03/98	13/03/98	18/03/98
4	03/04/98	18/03/98	23/03/98	26/03/98	31/03/98	30/04/98	13/04/98	16/04/98	22/04/98
5	03/04/98	18/03/98	23/03/98	26/03/98	31/03/98	29/04/98	06/04/98	14/04/98	20/04/98
6	31/03/98	11/03/98	16/03/98	19/03/98	25/03/98	03/04/98	20/03/98	24/03/98	27/03/98
7	08/05/98	17/04/98	22/04/98	27/04/98	01/05/98	02/04/98	16/03/98	19/03/98	26/03/98
8	27/03/98	09/03/98	13/03/98	18/03/98	24/03/98	27/03/98	11/03/98	16/03/98	19/03/98
9	27/03/98	10/03/98	13/03/98	18/03/98	24/03/98	03/04/98	13/03/98	20/03/98	25/03/98
10	30/04/98	13/04/98	16/04/98	22/04/98	27/04/98	03/04/98	18/03/98	23/03/98	26/03/98
11	29/04/98	06/04/98	14/04/98	20/04/98	23/04/98	01/05/98	17/04/98	21/04/98	24/04/98
12	03/04/98	20/03/98	24/03/98	27/03/98	01/04/98	24/03/98	27/03/98	01/04/98	06/04/98
13	02/04/98	16/03/98	19/03/98	26/03/98	30/03/98	19/03/98	26/03/98	30/03/98	20/03/98
14	27/03/98	11/03/98	16/03/98	19/03/98	24/03/98	16/03/98	19/03/98	24/03/98	16/03/98
15	03/04/98	13/03/98	20/03/98	25/03/98	31/03/98	20/03/98	25/03/98	31/03/98	11/03/98
16	03/04/98	18/03/98	23/03/98	26/03/98	31/03/98	23/03/98	26/03/98	31/03/98	13/03/98
17	01/05/98	17/04/98	21/04/98	24/04/98	28/04/98	21/04/98	24/04/98	28/04/98	18/03/98
18	03/04/98	12/03/98	18/03/98	24/03/98	30/03/98	18/03/98	24/03/98	30/03/98	17/04/98



Si l'utilisateur clique sur le bouton Annuler ou sur le bouton de fermeture d'une boîte de dialogue affichée à l'aide de la fonction `InputBox`, la fonction renvoie une chaîne vide. Une erreur pourra alors être générée par le programme s'il tente d'exploiter la valeur retournée. Placez une instruction `If...Then...Else` pour vérifier que la valeur retournée par `InputBox` n'est pas une chaîne vide. Les instructions suivantes pourront être placées sous la ligne 11 du programme précédent afin de mettre fin à la procédure si l'utilisateur annule la saisie d'une valeur.

```
If DateEcheance =    Then  
    Exit Sub  
End if
```

De la même façon, si l'utilisateur saisit une valeur ne correspondant pas à la valeur attendue, le programme pourra générer une erreur ou produire des résultats erronés. Utilisez les instructions de contrôle de type données présentées au Chapitre 6 pour vous assurer que les informations fournies par l'utilisateur sont valides. Dans le cas du programme précédent, si l'utilisateur ne saisit pas une date dans la boîte de dialogue, une erreur sera générée. Vous écrierez le code permettant de gérer ces éventuelles erreurs au Chapitre 10.

La méthode ***InputBox***

L'objet Application d'Excel possède une méthode `InputBox`, que vous pouvez substituer à la fonction `InputBox` de Visual Basic. L'intérêt de la méthode `InputBox` d'Excel est qu'elle permet de spécifier le type de données qui sera renvoyé. Utilisez cette méthode selon la syntaxe suivante :

```
Application.InputBox(prompt, title, default, left, top, helpFile, helpContextID,  
type)
```

Si vous ne spécifiez pas de valeur pour l'argument *title*, le titre par défaut de la boîte de dialogue sera "Entrée". Les arguments nommés *left* et *top* permettent de spécifier l'emplacement de la boîte de dialogue sur l'écran au moment de son affichage, tandis que les arguments *helpFile* et *helpContextID* servent à associer des fichiers d'aide à la boîte de dialogue. Ils existent aussi pour la fonction `InputBox` de Visual Basic.

L'argument de type Variant *type* est facultatif. Il peut prendre l'une des valeurs présentées dans le Tableau 7.2, qui déterminera le type de données renvoyé par la méthode `InputBox`. La méthode `InputBox` peut renvoyer plusieurs types de données définis. Affectez à l'argument *type* le résultat de la somme des valeurs correspondantes. Par exemple, pour que

l'utilisateur soit autorisé à indiquer une valeur numérique ou une référence de cellules, vous affecterez la valeur 9 à l'argument *type* (1 + 8).

Tableau 7.2 : Valeurs admises par l'argument *type* de la méthode *InputBox* d'Excel

<i>Valeur de Type</i>	<i>Type de données renvoyé par InputBox</i>
0	Formule
1	Valeur numérique
2	Chaîne de caractères
4	Valeur booléenne (False ou True)
8	Référence de cellule (objet Range)
16	Valeur d'erreur
64	Tableau de valeurs



Si l'information saisie par l'utilisateur dans la zone de texte de la boîte de dialogue ne correspond pas au type de données déclaré pour la méthode InputBox, une erreur sera générée. Pensez à créer un gestionnaire d'erreur (voir Chapitre 10), afin de prévoir les erreurs de saisie de l'utilisateur final.

L'autre avantage de la méthode InputBox d'Excel sur sa concurrente Visual Basic est de permettre à l'utilisateur de sélectionner une plage de cellules avant de cliquer sur le bouton OK. Il est ainsi possible de sélectionner un classeur ou une feuille spécifique, puis de sélectionner la plage voulue. Les coordonnées de celles-ci s'affichent alors dans la zone de texte de la boîte de dialogue. L'instruction suivante affiche une boîte de dialogue qui accepte pour valeur une référence de cellule. L'utilisateur est invité à sélectionner une plage de cellules dans la feuille active (voir Figure 7.15).

```
Sub RenvoyerUnePlageAvecInputBox()
    Dim MaPlage As Range
    Set MaPlage = Application.InputBox(prompt:="Sélectionnez la plage de cellules.", _
        Title:="Contrôle des échéances", Left:=3, Top:=-80, Type:=8)
End Sub
```

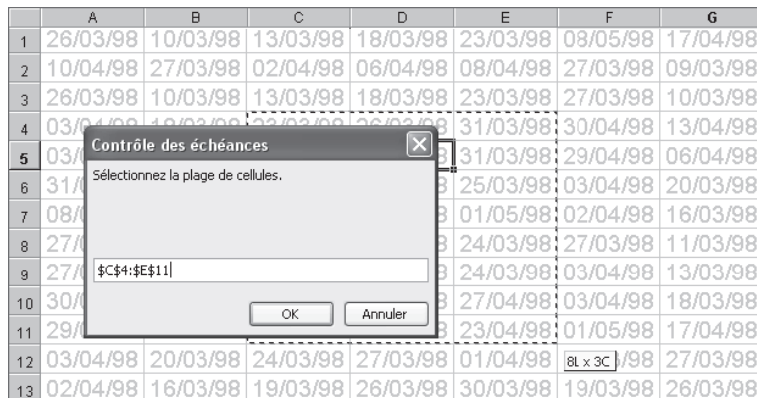
Entraînez-vous ! Modifiez la procédure *VerifierEcheances*, de façon à inviter l'utilisateur à sélectionner la plage de cellules à traiter, plutôt que de traiter la plage sélectionnée au moment de l'exécution de la macro.

Conseil

Lorsque vous utilisez la fonction `InputBox` d'Excel pour inviter l'utilisateur à sélectionner une plage de cellules, tirez profit des arguments `Left` et `Top`. Si la boîte de dialogue s'affiche dans l'angle supérieur gauche de la fenêtre, l'utilisateur n'aura pas besoin de la déplacer pour sélectionner des cellules sur la feuille.

Figure 7.15

La zone de texte de la boîte de dialogue reflète la sélection effectuée sur la feuille Excel.



La fonction `MsgBox`

La fonction `MsgBox` permet d'afficher une boîte de dialogue présentant un message et des boutons de commande, afin d'afficher une information à l'attention de l'utilisateur ou d'obtenir une réponse à une question qui orientera l'exécution du programme. Une valeur de type `Integer` est renvoyée en fonction du bouton cliqué et stockée dans une variable, pour être ensuite exploitée par le programme.

La fonction `MsgBox` s'utilise selon la syntaxe suivante :

```
Variable = MsgBox(prompt, buttons, title)
```

où *Variable* est une variable de type `Integer`. *prompt*, *buttons* et *title* sont des arguments nommés. *prompt* est un argument de type `String`, correspondant au message affiché dans la boîte de dialogue. *buttons* est un argument de type numérique facultatif. Il détermine les boutons affichés dans la boîte de dialogue, le symbole identifiant le type du message (information, question, etc.) et le bouton par défaut. Si cet argument est omis, un seul bouton libellé OK s'affiche et aucune icône n'identifie le type du message. *title* est un argument de type `String` facultatif, correspondant au titre de la boîte de dialogue (affiché dans la barre de titre). Si cet argument est omis, le nom de l'application apparaît dans la barre de titre.

L'argument *buttons* est défini par la somme des valeurs choisies pour chacun des groupes présentés dans le Tableau 7.3. Votre code gagnera cependant en lisibilité si vous utilisez les constantes VBA intégrées plutôt que des valeurs numériques.

Tableau 7.3 : Définition de l'argument *buttons*

<i>Constante</i>	<i>Valeur</i>	<i>Description</i>
Bouton		
<code>vbOKOnly</code>	0	OK
<code>vbOKCancel</code>	1	OK et Annuler
<code>vbAbortRetryIgnore</code>	2	Abandonner, Réessayer et Ignorer
<code>vbYesNoCancel</code>	3	Oui, Non et Annuler
<code>vbYesNo</code>	4	Oui et Non
<code>vbRetryCancel</code>	5	Réessayer et Annuler
Symbole		
<code>vbCritical</code>	16	Message critique
<code>vbQuestion</code>	32	Question
<code>vbExclamation</code>	48	Stop
<code>vbInformation</code>	64	Information
Bouton actif par défaut		
<code>vbDefaultButton1</code>	0	Premier bouton
<code>vbDefaultButton2</code>	256	Deuxième bouton
<code>vbDefaultButton3</code>	512	Troisième bouton
<code>vbDefaultButton4</code>	768	Quatrième bouton

Si vous souhaitez, par exemple, afficher une boîte de dialogue contenant le symbole "Question", dans laquelle l'utilisateur pourra répondre à la question posée par "Oui" ou "Non", l'argument *buttons* sera égal à 36 : 4 (pour les boutons) + 32 (pour le symbole). Les trois instructions suivantes sont équivalentes et entraînent l'affichage de la boîte de dialogue représentée à la Figure 7.16. La troisième formule est cependant plus compréhensible.

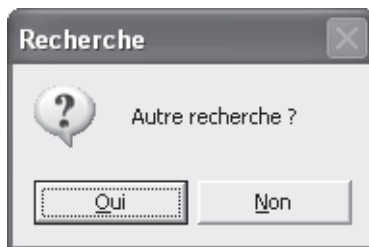
```

réponse = MsgBox("Autre recherche ?", 36, "Recherche")
réponse = MsgBox("Autre recherche ?", 32 + 4, "Recherche")
réponse = MsgBox("Autre recherche ?", vbYesNo + vbQuestion, "Recherche")

```

Figure 7.16

Une boîte de dialogue affichée à l'aide de l'instruction MsgBox.

**Astuce**

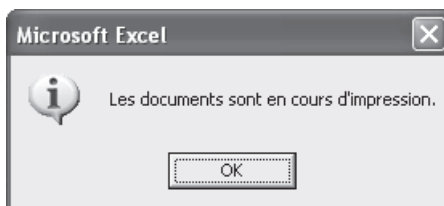
Lorsque vous insérez la fonction MsgBox dans une procédure, placez le curseur sur la fonction, puis tapez sur la touche F1 afin d'en activer la rubrique d'aide. Vous pourrez ainsi consulter le tableau répertoriant les constantes à attacher à l'argument buttons.

Si vous souhaitez afficher une boîte de dialogue dans le seul objectif d'afficher une information à l'attention de l'utilisateur (ne comportant qu'un bouton OK), il est inutile d'affecter une variable à la fonction MsgBox. Cependant, une fonction ne peut être utilisée que dans une instruction d'affectation. Vous devez alors utiliser l'instruction MsgBox. Sa syntaxe est la même que celle de sa sœur fonction, à la différence près que ses arguments ne sont pas encadrés par des parenthèses. L'instruction MsgBox suivante affiche la boîte de dialogue présentée à la Figure 7.17.

```
MsgBox "Les documents sont en cours d'impression.", vbInformation
```

Figure 7.17

Pour afficher un simple message informatif, utilisez une instruction plutôt qu'une fonction.



Par défaut, le premier bouton est le bouton actif (celui qui sera validé si l'utilisateur appuie sur la touche Entrée). Si une réponse positive de la part de l'utilisateur peut être lourde de conséquences, vous pouvez définir le deuxième ou le troisième bouton comme bouton par défaut. L'instruction suivante affiche la boîte de dialogue présentée à la Figure 7.18 :

```
réponse = MsgBox("Supprimer toutes les informations ?", _  
vbYesNoCancel + vbCritical + vbDefaultButton2)
```

Figure 7.18

Si l'utilisateur appuie sur la touche Entrée du clavier, c'est la réponse "Non" qui sera validée.



La valeur renvoyée par la fonction MsgBox varie en fonction du bouton cliqué par l'utilisateur, selon la règle suivante :

<i>Bouton</i>	<i>Constante</i>	<i>Valeur</i>
OK	vbOK	1
Annuler	vbCancel	2
Abandonner	vbAbort	3
Réessayer	vbRetry	4
Ignorer	vbIgnore	5
Oui	vbYes	6
Non	vbNo	7

Une instruction conditionnelle est utilisée pour interroger la valeur de la variable recevant l'information et orienter le déroulement du programme en conséquence. Là encore, vous pouvez interroger la valeur de la variable en utilisant la constante VBA ou la valeur numérique correspondante. Dans le cas de la boîte de dialogue présentée à la Figure 7.18, si l'utilisateur clique sur le bouton Oui, la valeur 6, correspondant à la constante vbYes, sera affectée à la variable réponse ; s'il clique sur le bouton Non, la valeur 7, comparable à la constante vbNo sera retournée ; enfin, s'il choisit le bouton Annuler c'est la valeur 2, assimilable à la constante vbCancel, qui sera renvoyée.

Le programme suivant reprend l'exemple donné précédemment pour InputBox. Une fonction MsgBox a été utilisée afin de proposer à l'utilisateur de définir une autre date, en cas de dépassement d'échéance. S'il n'y a pas de dépassement d'échéance, une autre boîte de dialogue sera affichée pour en informer l'utilisateur.

```

1: Sub VerifierEcheances()
2:   Dim ZoneATester As String
3:   ZoneATester = ActiveWindow.RangeSelection.Address
4:   If ZoneATester = Null Then
5:     MsgBox "Sélectionnez la plage de cellules à tester.", _
        vbOKOnly & vbInformation

```

```

6:      Exit Sub
7:  End If

8:  Dim DateEcheance As Variant
9:  DateEcheance = InputBox("Indiquez la date d'échéance.", _
    "Echéance des opérations en cours", Date + 30)
10:  DateEcheance = CDate(DateEcheance)

11:  Dim CellTest As Range
12:  Dim DatesHorsEcheance As Boolean
13:  DatesHorsEcheance = False
14:  For Each CellTest In Range(ZoneATester)
15:      If IsDate(CellTest) = True Then
16:          If CellTest.Value > DateEcheance Then
17:              CellTest.Interior.ColorIndex = 6
18:              DatesHorsEcheance = True
19:          End If
20:      End If
21:  Next

22:  If DatesHorsEcheance = True Then
23:      Dim RedéfinirLaDate As Integer
24:      RedéfinirLaDate = MsgBox("Des problèmes d'échéance ont été trouvés." & _
Chr(10) & "Souhaitez-vous spécifier une autre date ?", _
vbYesNo + vbQuestion, "Redéfinir la date d'échéance ?")
25:      If RedéfinirLaDate = vbYes Then
26:          Range(ZoneATester).Interior.ColorIndex = xlNone
27:          Call VerifierEcheances
28:      End If
29:  Else
29:      MsgBox "Pas de problème d'échéance.", _
vbOKOnly + vbInformation, "Echéancier respecté"
30:  End If
31: End Sub

```

Lignes 12 et 13, la variable `DatesHorsEcheance` de type Boolean est déclarée et se voit affecter la valeur `False`. Ligne 18, une instruction lui affecte la valeur `True`. Cette instruction ne sera exécutée que si une date supérieure à la date spécifiée par l'utilisateur est trouvée.

La mise en valeur des éventuelles cellules hors échéance terminée, une instruction conditionnelle `If...Then...Else` est utilisée pour afficher un message à l'attention de l'utilisateur (lignes 22 à 30). Si la variable `DateHorsEcheance` renvoie la valeur `True` (si des cellules contenant des dates au-delà de la date spécifiée par l'utilisateur sont trouvées),

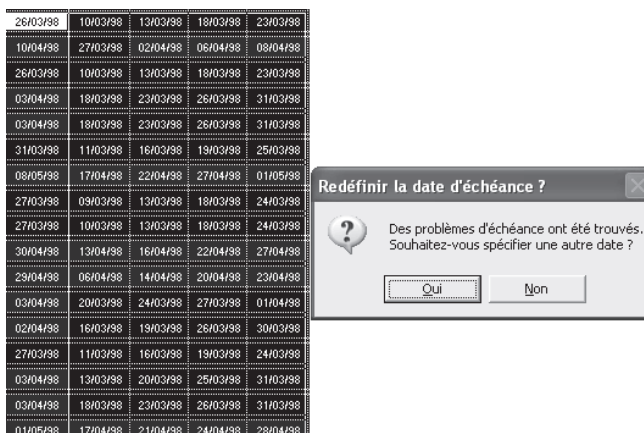
les instructions des lignes 23 à 28 sont exécutées. La boîte de dialogue représentée à la Figure 7.19 est alors affichée. La valeur renvoyée par la fonction MsgBox est stockée dans la variable RedéfinirLaDate. Une structure conditionnelle imbriquée teste ensuite la valeur de cette variable. Si la variable renvoie vbYes (l'utilisateur a cliqué sur le bouton Oui), la couleur d'intérieur des cellules de la zone sélectionnée est supprimée (ligne 26), et la fonction s'appelle elle-même (ligne 27). Si DateHorsEcheance renvoie False, une boîte de dialogue s'affiche (ligne 29) afin d'informer l'utilisateur qu'il n'y a pas de problème d'échéance.



Une procédure qui s'appelle elle-même est une procédure récursive.

Figure 7.19

L'utilisateur peut redéfinir une date d'échéance.



Affichage de boîtes de dialogue Excel

Il peut être utile d'afficher des boîtes de dialogue Excel à un moment spécifique de l'exécution d'un programme. Vous pouvez, par exemple, afficher la boîte de dialogue Enregistrer sous, afin que l'utilisateur indique le dossier d'enregistrement d'un classeur.

Les boîtes de dialogue prédéfinies d'Excel

Les boîtes de dialogue sont des objets Dialog. Pour afficher une boîte de dialogue Excel, faites appel à la collection Dialogs et appliquez la méthode Show ou Display à l'objet défini selon la syntaxe suivante :

```
Application.Dialogs(xlDialog).Show
```

```
Application.Dialogs(xlDialog).Display
```

`xlDialog` est une constante Excel qui indique la boîte de dialogue devant être affichée. L'instruction suivante affiche la boîte de dialogue Ouvrir :

```
Application.Dialogs(xlDialogOpen).Show
```

Lorsqu'une boîte de dialogue Excel s'affiche à l'aide de la méthode `Show`, les commandes qu'elle prend en charge s'exécutent normalement si l'utilisateur valide les paramètres qu'il y a définis. La procédure se poursuit ensuite avec l'instruction située derrière celle qui a appelé la boîte de dialogue.

Vous pouvez évidemment affecter un objet `Dialog` à une variable de type `Object` ou `Variant`, et faire appel à cet objet pour afficher la boîte en question. Les instructions suivantes affichent la boîte de dialogue Enregistrer sous.

```
Dim BoîteEnregistrerSous as Dialog  
Set BoîteEnregistrerSous = Application.Dialogs(xlDialogSaveAs)  
BoîteEnregistrerSous.Show
```

Vous trouverez une liste des constantes `xlDialog` dans l'aide de VBA pour Excel. Référez-vous à la rubrique Références pour Microsoft Excel/Objets/Dialogs.

Les méthodes *GetOpenFilename* et *GetSaveAsFilename*

Les méthodes `GetOpenFilename` et `GetSaveAsFilename` d'Excel permettent d'afficher les boîtes de dialogue standard Ouvrir et Enregistrer sous afin de récupérer le nom d'un fichier. Celles-ci sont plus souples et plus fonctionnelles que les objets `Dialogs` correspondants.



Contrairement à ce qui se passe lorsque vous appliquez la méthode `Show` à un objet `Dialog`, quand une boîte de dialogue s'affiche à l'aide de l'une des méthodes `GetOpenFilename` ou `GetSaveAsFilename`, aucune action n'est exécutée lorsque l'utilisateur clique sur le bouton de validation de la boîte de dialogue. Ces méthodes permettent simplement de récupérer un nom de fichier. Ce fichier doit ensuite être manipulé par programmation.

Utilisez les méthodes `GetOpenFilename` et `GetSaveAsFilename`, selon la syntaxe suivante :

```
Application.GetSaveAsFilename(InitialFilename, FileFilter, FilterIndex, Title,  
ButtonText)  
expression.GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSe-  
lect)
```

Tableau 7.4 : Arguments des méthodes *GetOpenFilename* et *GetSaveAsFilename*

Argument	Description
InitialFilename	Argument facultatif de type <code>Variant</code> . Le nom de fichier par défaut apparaissant dans la zone de texte Nom. Si cet argument est omis, le nom du classeur actif est utilisé comme nom par défaut.
FileFilter	Argument facultatif de type <code>Variant</code>. Les critères de filtrage des fichiers. Chaque critère doit apparaître sous la forme d'une paire composée du filtre de fichier, suivi de la spécification de son extension, tels qu'ils apparaissent dans la zone Type de fichier. Les paires sont également séparées par des virgules. Si plusieurs extensions sont associées à un type de fichier, elles doivent être séparées par un point-virgule. Par exemple la valeur suivante : Classeur Microsoft Excel (*.xlsx),*.xlsx,PageWeb (*.htm;*.html),*.htm;*.html" pour l'argument <code>FileFilter</code> laissera apparaître deux types de fichiers dans la zone Type de fichier de la boîte de dialogue affichée.
FilterIndex	Argument facultatif de type <code>Variant</code>. L'index du critère de filtrage par défaut. Spécifiez la valeur 1 pour que le premier critère de filtrage apparaisse comme type de fichier par défaut, la valeur 2 pour que le deuxième critère de filtrage soit le type de fichier par défaut, etc. Le premier filtre de fichier est utilisé si l'argument n'a pas été spécifié ou s'il est supérieur au nombre de filtres disponibles.
Title	Argument facultatif de type <code>Variant</code> . Le titre apparaissant dans la barre de titre de la boîte de dialogue. Si cet argument est omis, le titre de la boîte de dialogue est Ouvrir.
ButtonText	Argument facultatif de type <code>Variant</code> . Sur Macintosh uniquement. Le libellé du bouton Ouvrir.
MultiSelect	Argument facultatif de type <code>Variant</code>. Si <code>MultiSelect</code> a la valeur <code>True</code>, l'utilisateur peut effectuer une sélection multiple. Si <code>MultiSelect</code> a la valeur <code>False</code> (valeur par défaut), l'utilisateur ne peut sélectionner qu'un fichier. Si plusieurs fichiers sont sélectionnés, la valeur est renvoyée sous forme de tableau et doit donc être stockée dans une variable de type <code>Variant</code> ou <code>Array</code>.

Utilisez l'instruction `ChDir` pour modifier le dossier proposé par défaut dans la boîte de dialogue affichée.

L'exemple suivant affiche la boîte de dialogue représentée à la Figure 7.20, puis affiche une boîte de dialogue indiquant la liste des fichiers choisis par l'utilisateur (voir Figure 7.21).

```

1: Sub OuvertureDeFichiers()
2:   Dim OuvrirFichiers As Variant
3:   'modification du chemin par défaut
4:   ChDir ("c:\Documents and settings\Administrateur\bureau\")

```

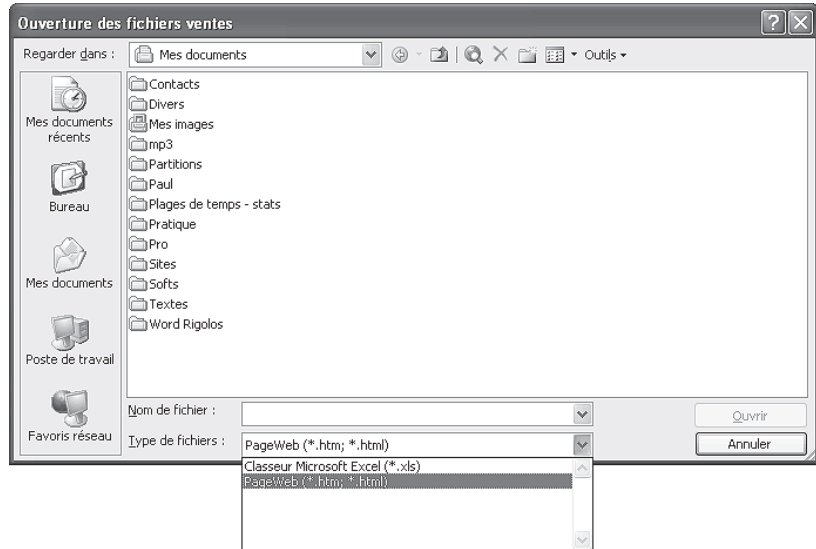
```
5: 'affichage de la boîte de dialogue Ouvrir
6: OuvrirFichiers = Application.GetOpenFilename(filefilter:="Classeur Microsoft
Excel (*.xlsx),*.xlsx,PageWeb (*.htm; *.html),*.htm;*.html", filterindex:=2,
Title:="Ouverture des fichiers ventes", MultiSelect:=True)
7: 'si l'utilisateur a sélectionné plusieurs fichiers
8: If UBound(OuvrirFichiers) > 1 Then
9:     Dim rep As Long
10:    Dim Liste As String
11:    Dim compteur As Byte
12:    For compteur = 1 To UBound(OuvrirFichiers)
13:        Liste = Liste & vbCrLf & OuvrirFichiers(compteur)
14:    Next compteur
15:    'affichage de l'ensemble de la liste des fichiers et proposition
    d'ouverture
16:    rep = MsgBox("L'utilisateur a sélectionné plusieurs fichiers. En voici
la liste." & _
17:    Liste & vbCrLf & "Voulez-vous les ouvrir ?", vbYesNo + vbQuestion, _
18:    "Ouvrir les fichiers ?")
19:
20:    'ouverture des fichiers en cas de réponse positive
21:    If rep = vbYes Then
22:        For compteur = 1 To UBound(OuvrirFichiers)
23:            Workbooks.Open Filename:=OuvrirFichiers(compteur)
24:        Next compteur
25:    End If
26:    'si un seul fichier a été sélectionné, il est ouvert
27:    Else
28:        Workbooks.Open Filename:=OuvrirFichiers(1)
29:    End If
30: End Sub
```

La variable `OuvrirFichiers` déclarée ligne 2 servira à stocker le(s) fichier(s) sélectionné(s) par l'utilisateur. Ligne 4, le chemin par défaut est modifié à l'aide de l'instruction `ChDir`, de façon à ce que le dossier proposé par défaut lors de l'affichage de la boîte de dialogue soit le Bureau Windows.

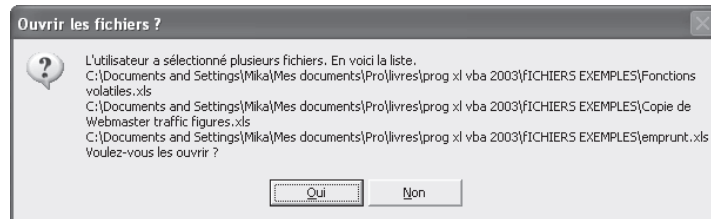
Ligne 6, la boîte de dialogue `Ouvrir` s'affiche. Le résultat de la sélection opérée par l'utilisateur est affecté à la variable `OuvrirFichiers`. L'argument `filefilter` est défini de façon que les types de fichiers apparaissant dans la liste Type de fichier soient les fichiers portant l'extension `.xlsx` et `.htm` ou `.html`. L'argument `filterindex` définit le deuxième type de fichier (`.htm` et `.html`) comme type par défaut. Enfin, le titre de la boîte de dialogue est défini à `Ouverture des fichiers ventes` et la sélection multiple est autorisée.

Figure 7.20

La méthode GetOpen-FileName permet d'afficher une boîte de dialogue Ouvrir personnalisée.

**Figure 7.21**

Les fichiers sélectionnés sont récupérés dans une variable de matrice.



Lignes 8 à 29, une instruction conditionnelle If...Then...Else sert à vérifier si plusieurs fichiers ont été sélectionnés. Si tel est le cas (la condition posée ligne 8 est vérifiée), les instructions des lignes 8 à 26 s'exécutent.

Lignes 9 à 11, les variables rep, Liste et Compteur sont déclarées. Lignes 12 à 14, une instruction For...Each...Next sert à stocker la liste des fichiers sélectionnés dans la variable Liste. La valeur initiale du compteur est 1, et elle est incrémentée de 1 à chaque passage de la boucle jusqu'à atteindre la valeur limite de la variable de matrice [UBound(OuvrirFichiers)]. À chaque passage, la variable Liste reçoit sa valeur + un retour chariot (& vbCr) + la valeur stockée dans la variable OuvrirFichiers, à la position correspondant à la valeur de Compteur [OuvrirFichiers(compteur)]. Les lignes 16 à 18, une instruction MsgBox est utilisée pour afficher la liste des fichiers sélectionnés par l'utilisateur et lui proposer de les ouvrir (voir Figure 7.21). Lignes 21 à 25, une instruction conditionnelle imbriquée sert à évaluer la réponse de l'utilisateur. Une instruction For...Next ouvre les fichiers en cas de réponse positive. Si un seul fichier a été sélectionné (ligne 27), il est ouvert (ligne 28).



*Si l'un des classeurs sélectionnés ne s'affiche pas, c'est qu'il est masqué. Utilisez la commande **Afficher** pour en forcer l'affichage. Si vous utilisez Excel 2007, cette fonction est accessible via l'onglet **Affichage** du ruban ; avec une version ultérieure, utilisez la commande **Afficher** du menu **Fenêtre**.*

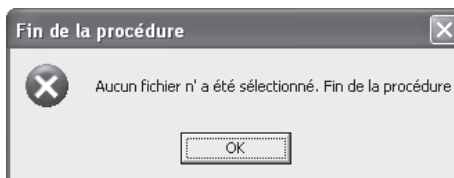
Tel qu'il se présente ici, ce programme générera une erreur si l'utilisateur clique sur le bouton **Annuler** ou sur la croix de fermeture de la boîte de dialogue. Ajoutez les instructions apparaissant en gras dans le programme suivant pour résoudre ce problème. Si aucun fichier n'est sélectionné, le message de la Figure 7.22 s'affiche et la procédure prend fin.

```
Sub OuvertureDeFichiers()
    Dim OuvrirFichiers As Variant
    ChDir ("c:\Documents and settings\Administrateur\bureau\")
    OuvrirFichiers = Application.GetOpenFilename(filefilter:="Classeur Microsoft
Excel (*.xlsx),*.xlsx,PageWeb (*.htm; *.html),*.htm;*.html", filterindex:=2,
Title:="Ouverture des fichiers ventes", MultiSelect:=True)
    If OuvrirFichiers = False Then
        MsgBox "Aucun fichier n' a été sélectionné. Fin de la procédure", _
            VbOKOnly + vbCritical, "Fin de la procédure"
        Exit Sub
    End If
    If UBound(OuvrirFichiers) > 1 Then
        Dim rep As Long
        Dim Liste As String
        Dim compteur As Byte
        For compteur = 1 To UBound(OuvrirFichiers)
            Liste = Liste & vbCr & OuvrirFichiers(compteur)
        Next compteur
        rep = MsgBox("L'utilisateur a sélectionné plusieurs fichiers. En voici la
liste." & _
            Liste & vbCr & "Voulez-vous les ouvrir ?", vbYesNo + vbQuestion, _
            "Ouvrir les fichiers ?")

        If rep = vbYes Then
            For compteur = 1 To UBound(OuvrirFichiers)
                Workbooks.Open Filename:=OuvrirFichiers(compteur)
            Next compteur
        End If
    Else
        Workbooks.Open Filename:=OuvrirFichiers(1)
    End If
End Sub
```

Figure 7.22

L'annulation de l'ouverture est maintenant gérée par le programme.



Utiliser les opérateurs logiques

Visual Basic propose des opérateurs logiques, qui permettront d'optimiser les instructions conditionnelles `While...Wend` et `If...Then...Else`. Ils permettent en effet de combiner plusieurs conditions et, ainsi, de simplifier les instructions conditionnelles dans bien des cas. L'instruction conditionnelle suivante utilise l'opérateur logique `And` pour vérifier que la valeur de la cellule A3 est inférieure à 1 000 et supérieure à 2 000 :

```
If Range("A3").Value < 1000 and Range("A3").Value > 2000 Then...
```

Les conditions établies à l'aide d'opérateurs logiques sont proches d'une condition exprimée dans un langage courant, ce qui en facilite considérablement la compréhension.

Les instructions disponibles pour utiliser les opérateurs logiques sont présentées dans le Tableau 7.5.

Tableau 7.5 : Les opérateurs logiques

<i>Opérateur</i>	<i>Description</i>
Or	Contrôle que l'une des conditions spécifiées est vérifiée. Si c'est le cas, la condition renvoie <code>True</code> .
And	Contrôle que toutes les conditions spécifiées sont vérifiées. Si c'est le cas, l'expression conditionnelle renvoie <code>True</code> .
Xor	Ou restrictif. Si l'une ou l'autre des conditions est respectée, l'expression conditionnelle renvoie <code>True</code> . Si plus d'une condition est vérifiée, l'expression conditionnelle renvoie <code>False</code> .
Not	Cet opérateur nie l'expression devant laquelle il est situé. La structure conditionnelle <code>If Not condition Then...</code> est vérifiée si la <i>condition</i> n'est pas respectée.

Astuce

Vous pouvez employer les opérateurs `Or` et `And` un nombre de fois indéterminé dans une même expression. Cependant, pour combiner ces opérateurs dans une

même expression, vous devez utiliser des parenthèses, afin de spécifier quels sont les rapports entretenus entre les différentes conditions. Vous obtiendrez alors des expressions du type :

```
If (Condition1 Or Condition2) And Condition3 Then  
Série d'instructions  
End If
```

Dans le cas d'une telle expression, la Série d'instructions ne sera exécutée que si l'une des deux premières conditions (ou les deux) est vérifiée, et si la Condition3 est aussi vérifiée.

Trier des données

Trier et rechercher des données est une opération courante pour l'utilisateur d'Excel. La recherche étant en général subordonnée au tri des données, cette question est fondamentale pour le développeur d'applications pour Excel. Un programme VBA peut trier une zone définie d'une feuille Excel comme un utilisateur le ferait *via* la commande Trier du menu Données, en faisant appel à la méthode Sort.

Vous serez cependant amené à trier des données stockées dans une variable et non sur une feuille Excel. Par ailleurs, si un programme effectue de nombreuses opérations d'analyse, de traitement et de tri sur des données, vous gagnerez en rapidité d'exécution en travaillant sur ces données en mémoire plutôt qu'à même la feuille Excel. La méthode la plus simple consiste alors à stocker les données de la feuille dans une variable tableau et à travailler sur cette variable.

Les techniques de tri des variables tableau sont nombreuses. Nous vous proposons ici une méthode simple et efficace qui consiste à comparer des éléments adjacents dans le tableau et à les intervertir si nécessaire. On utilise pour cela une boucle For...Next imbriquée. Considérez la fonction suivante :

```
1: Public Function TriTableau(MaVar())  
2:     Dim i As Long  
3:     Dim j As Long  
4:     Dim temp  
5:     For i = LBound(MaVar) To UBound(MaVar) - 1  
6:         For j = i + 1 To UBound(MaVar)  
7:             If MaVar(i) > MaVar(j) Then  
8:                 temp = MaVar(i)  
9:                 MaVar(i) = MaVar(j)  
10:                MaVar(j) = MaVar(i)  
11:            End If
```

```
12:         Next j
13:     Next i
14:     TriTableau = MaVar
15: End Function
```

L’instruction de déclaration de notre fonction indique qu’elle attend la variable de tableau `MaVar` en argument. Lignes 2 à 4, les variables sont déclarées. `i` et `j` sont les compteurs de nos boucles `For...Next`, tandis que la variable `temp` servira simplement à stocker provisoirement des données.

Lignes 5 à 13 se trouvent les boucles `For...Next` imbriquées qui assurent le tri de notre variable. La boucle principale s’exécute autant de fois que la variable contient d’espaces de stockage – 1. Elle permet ainsi d’effectuer un tri de la variable indice par indice, du premier au dernier.

La boucle imbriquée (lignes 6 à 12) effectue le tri à proprement parler. La valeur traitée est comparée à toutes les valeurs qui sont stockées après elle dans la variable tableau (`j = i + 1 To UBound(MaVar)`). Lors de chaque comparaison, si la valeur traitée est supérieure à la valeur à laquelle elle est comparée (ligne 7), les deux valeurs sont interverties (lignes 8 à 10). On stocke pour cela la valeur de `mavar(i)` dans la variable `temp`, puis on lui substitue la valeur de `mavar(j)`. On remplace ensuite la valeur de `mavar(j)` par la valeur de `mavar(i)` précédemment mémorisée dans la variable `temp`.

Une fois la boucle imbriquée exécutée, la valeur stockée à l’indice `i` de notre variable est plus petite que toutes celles qui la suivent. La structure `For...Next` principale reprend alors la main et passe à l’indice suivant de la variable tableau. Une fois l’opération menée pour l’ensemble des indices de la variable, les valeurs sont stockées par ordre croissant. Notre fonction reçoit alors la valeur de `MaVar` (ligne 14) et prend fin, retournant la variable maintenant triée.



Pour un tri décroissant, remplacez simplement le signe > de la ligne 7 par le signe <.

Dans l’exemple suivant, on utilise la même méthode de comparaison pour trier les feuilles de travail du classeur actif. Cette fois-ci les données ne sont pas stockées dans une variable tableau, mais les feuilles sont directement triées sur le classeur.

```
1: Sub TriFeuilles()
2:     Dim I As Integer
3:     Dim J As Integer
4:     Dim Min As Integer
5:     Dim ModeCalcul As Integer
6:     ModeCalcul = Application.Calculation
```

```
7: Application.Calculation = xlCalculationManual
8: Application.ScreenUpdating = False
9: With ActiveWorkbook.Worksheets
10:     For I = 1 To .Count - 1
11:         Min = I
12:         For J = I + 1 To .Count
13:             If .Item(J).Name < .Item(Min).Name Then Min = J
14:         Next J
15:         If Min <> I Then .Item(Min).Move before:=Worksheets(I)
16:     Next I
17: End With
18: Application.Calculation = ModeCalcul
19: Application.ScreenUpdating = True
20: End Sub
```

Lignes 2 à 5, les variables sont déclarées. Ligne 6, la valeur `Application.Calculation` (le mode de calcul défini pour le classeur) est mémorisée. Les lignes 7 et 8 sont destinées à optimiser le temps d'exécution de la macro. Le classeur est ainsi passé en mode de calcul manuel ligne 7 afin d'empêcher un éventuel recalcul des valeurs des cellules à chaque déplacement d'une feuille. Ligne 8, la valeur `False` est affectée à la propriété `ScreenUpdating` de l'objet `Application` afin de ne pas mettre à jour l'affichage à l'écran lors de l'exécution de la macro.

Lignes 9 à 17, une structure `With...End With` est utilisée avec l'objet `ActiveWorkbook.Worksheets` afin de simplifier l'écriture du code. Lignes 10 à 16, deux boucles `For...Next` imbriquées assurent le tri des feuilles.

La première boucle s'exécute autant de fois qu'il y a de feuilles dans le classeur - 1. À chaque passage, la variable `Min` reçoit pour valeur l'index de la feuille en cours de traitement. La boucle imbriquée (lignes 12 à 14) compare alors le nom stocké dans la variable `Min` avec le nom de chaque feuille située après elle. À chaque comparaison, si un nom inférieur (alphabétiquement antérieur) est trouvé, la variable `Min` reçoit pour valeur l'index de la feuille portant ce nom.

À la fin de cette boucle, la variable `Min` contient donc la valeur d'index de la feuille portant le nom le plus petit parmi celles qui ont été comparées. Ligne 15, on vérifie si on a trouvé une feuille portant un nom plus petit lors des comparaisons (`If Min <> I`). Si c'est le cas, cette feuille est placée devant la feuille en cours de traitement (`.Item(Min).Move before:=Worksheets(I)`).

La boucle principale reprend alors la main et la feuille suivante est traitée selon le même principe. Une fois les feuilles du classeur triées par ordre croissant de nom, le mode de calcul du classeur est redéfini à son état initial (ligne 18) et la mise à jour de l'affichage écran est réactivée (ligne 19).

Conseil

Notez que, dans l'exemple précédent, on évite des opérations inutiles de déplacement de feuilles en mémorisant la donnée la plus petite lors des comparaisons (ligne 13) et en n'effectuant qu'une seule substitution de position si nécessaire à la fin du passage de la boucle imbriquée (ligne 15). Exercez-vous à modifier la fonction `TriTableau` présentée plus avant selon le même principe.



8

Fonctions Excel et VBA

Au sommaire de ce chapitre

- Utiliser les fonctions Excel dans VBA
- Créer des fonctions Excel personnalisées
- Intégrer une fonction *via* l'Explorateur d'objets
- Recommandations pour l'écriture de fonctions Excel
- Principales fonctions VBA

Si les fonctions sont un élément clé du développement d'applications, quels que soient le langage et l'environnement de programmation, c'est particulièrement vrai pour Excel, dont l'activité principale consiste à effectuer des calculs.

Vos projets VBA peuvent ainsi exploiter des fonctions Excel comme des fonctions VBA, mais vous pouvez également créer vos propres fonctions afin de compléter celles qui sont proposées par défaut dans le tableur.

Utiliser les fonctions Excel dans VBA

Excel offre un nombre impressionnant de fonctions intégrées. De la simple addition aux fonctions de calculs avancés, les fonctions d'Excel permettent de réaliser nombre d'opérations. La plupart de ces fonctions peuvent être manipulées *via* du code Visual Basic.



Ne confondez pas les fonctions spécifiques à Excel et les fonctions Visual Basic. Si de nombreuses fonctions mathématiques Excel ont leur équivalent Visual Basic, Excel propose des fonctions avancées (statistiques, financières, etc.) qui sont bel et bien des fonctions Excel et non Visual Basic. Un programme exploitant ces fonctions ne pourra donc pas être exécuté dans une application hôte autre qu'Excel.

Pour utiliser une fonction Excel dans un programme VBA, vous ferez appel à l'objet `Application` qui représente l'application hôte (en l'occurrence Excel) qui contient ces fonctions. Dans l'exemple suivant, la fonction `QueEstLeMax` exploite la fonction `Max` d'Excel qui renvoie la valeur la plus forte dans une liste d'arguments. Si vous ne faites pas précéder la fonction `Max` du mot clé `Application`, la fonction ne sera pas reconnue et une erreur sera générée.

```
Public Sub QueEstLeMax()  
    MsgBox "Le plus grand des chiffres reçus est " & _  
        & Application.Max(1, 2, 3, 4), vbInformation + vbOKOnly, _  
        "Utiliser les fonctions Excel dans VB"  
End Sub
```

Créer des fonctions Excel personnalisées

Vous serez probablement amené à utiliser des fonctions qu'Excel ne prendra pas en charge. Le développement de procédures de type `Function` permet de remédier à ce problème.

Les fonctions personnalisées que vous créez dans Visual Basic Editor peuvent être utilisées dans Excel comme n'importe quelle fonction intégrée. Elles apparaissent alors dans la liste des fonctions d'Excel et peuvent être employées dans un classeur Excel à l'aide de la commande Insérer une fonction du menu Formule. Créez la fonction suivante dans Visual Basic Editor :

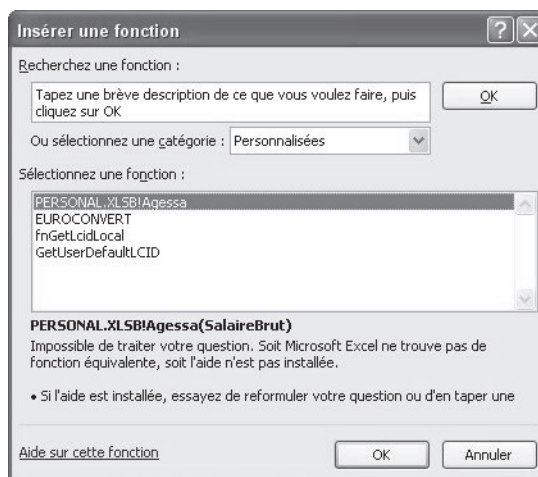
```
Function Agessa(SalaireBrut)
    Agessa = (SalaireBrut * 0.95) * 0.0085
End Function
```

Cette fonction calcule les AGESEA (cotisation sur les droits d'auteur). Les AGESEA se calculent sur une base de 95 % du salaire brut, au taux de 0,85 %.

La fonction Agessa étant créée, retournez dans Excel. Choisissez la commande Insérer une fonction du menu Formule et sélectionnez la catégorie Personnalisées. La fonction apparaît dans la liste Sélectionnez une fonction, précédée du nom du classeur dans lequel elle est stockée. Les arguments sont aussi visibles (voir Figure 8.1).

Figure 8.1

Les fonctions créées dans Visual Basic Editor peuvent être exploitées dans Excel, comme des fonctions intégrées.

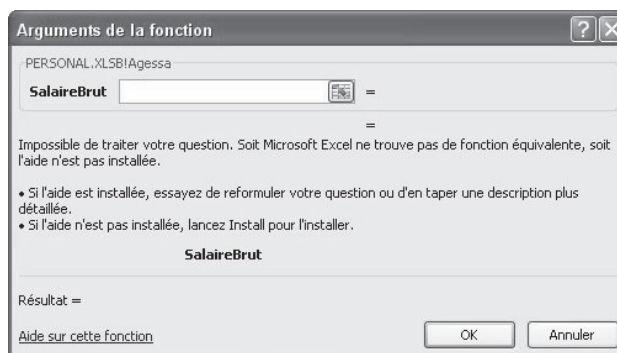


Si vous utilisez une version d'Excel antérieure à 2007, choisissez Insertion puis Fonction pour afficher la boîte de dialogue présentée à la figure 8.1.

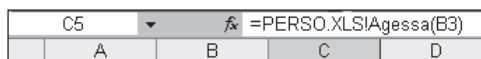
Les fonctions créées dans Visual Basic Editor sont gérées dans Excel comme n'importe quelle fonction intégrée du tableur. Si vous cliquez sur le bouton OK de la boîte de dialogue Insérer une fonction, vous serez invité à préciser les cellules correspondant aux arguments de la fonction (voir Figure 8.2). La fonction apparaîtra ensuite dans la barre de formule.

Figure 8.2

Vous êtes invité à spécifier les cellules correspondant aux arguments de la fonction.

**Figure 8.3**

La fonction personnalisée s'affiche dans la barre de formule.



Affectez des noms représentatifs aux arguments des fonctions que vous créez, afin qu'ils soient compréhensibles pour les autres utilisateurs.

Intégrer une fonction *via* l'Explorateur d'objets

Au vu des centaines de fonctions proposées par Excel et accessibles dans VBA, il est difficile de mémoriser la syntaxe et les arguments de l'ensemble de ces fonctions. L'Explorateur d'objets est alors la solution. Il recense en effet les fonctions Excel et VBA – intégrées comme personnalisées – et en détaille l'usage et la syntaxe.

Insérer une fonction VBA dans votre code

Pour insérer une fonction VBA dans votre code à partir de l'Explorateur d'objets, procédez comme suit :

1. Lancez l'Explorateur d'objets.
2. Dans la liste déroulante Projet/Bibliothèque, choisissez VBA.

La zone Classes affiche la liste des objets, fonctions, procédures et constantes de VBA.

3. Dans la zone Classes, sélectionnez la catégorie de fonctions voulue (Conversions, DateTime, Financial, Math, Strings, etc.).
4. Dans la zone Membres de, sélectionnez la fonction voulue.

Les détails de la fonction apparaissent dans la zone inférieure de l'Explorateur d'objets. Si vous souhaitez consulter l'aide en ligne pour la fonction choisie, cliquez sur le bouton Aide.

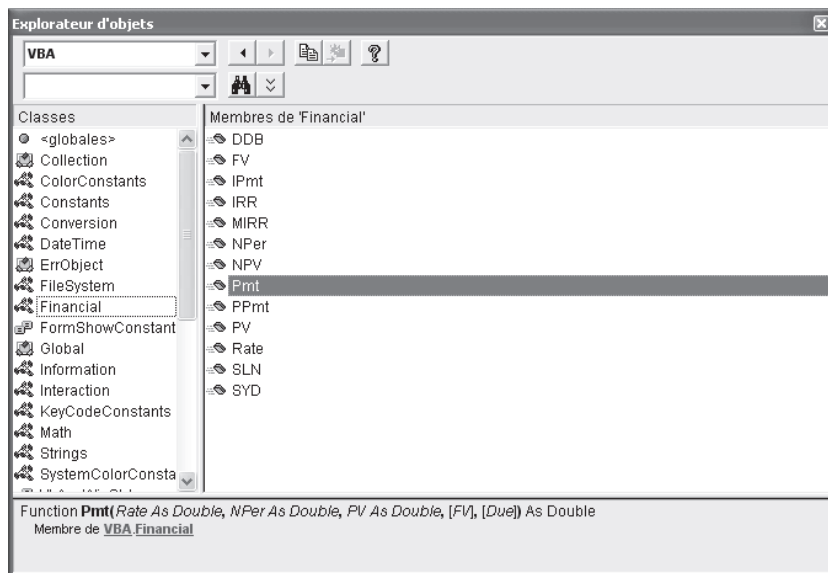


Figure 8.4

L'Explorateur d'objets permet d'accéder à l'ensemble des informations disponibles pour une fonction.

5. Cliquez ensuite sur le bouton Copier dans le Presse-papiers, puis fermez l'Explorateur d'objets.
6. Dans la fenêtre Code, placez le curseur à l'endroit souhaité, puis tapez le raccourci Ctrl+V afin de coller la fonction.

La fonction apparaît dans le code. Lorsque vous saisissez un espace, la liste des arguments attendus s'affiche.

Insérer une fonction Excel dans votre code

Pour insérer une fonction Excel dans votre code, procédez comme suit :

1. Lancez l'Explorateur d'objets.

2. Dans la liste Projet/Bibliothèque, choisissez Excel. Dans la zone Classes, choisissez WorksheetFunction.

Les fonctions Excel disponibles dans VBA apparaissent dans la zone Membres de.

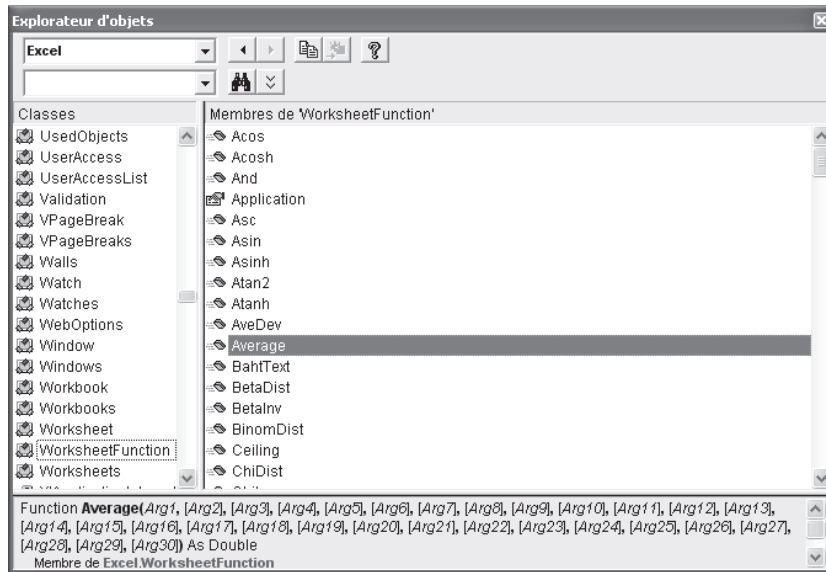


Figure 8.5

Les fonctions Excel sont accessibles via l'objet WorksheetFunction.

3. Sélectionnez la fonction voulue, puis copiez-la.
4. Placez ensuite le curseur à l'endroit où vous souhaitez faire apparaître la fonction dans la fenêtre Code.
5. Saisissez Application.WorksheetFunction., puis collez la fonction.

Astuce

Plutôt que de passer par l'Explorateur d'objets, vous pouvez simplement saisir Application.WorksheetFunction. dans votre code pour faire apparaître la liste des fonctions disponibles.

Notez que vous pouvez omettre la propriété WorksheetFunction et vous contenter de faire précéder la fonction par la propriété Application. Le code fonctionnera également, mais VBA n'affichera pas d'aide à l'écriture de code lors de la saisie dans la fenêtre Code.

Recommandations pour l'écriture de fonctions Excel

Les limites de la cellule

N'oubliez pas que vos fonctions serviront en général à insérer des données dans une cellule. Elles doivent donc présenter les mêmes limites que celles qui s'appliquent aux cellules. Veillez donc à ce que vos fonctions ne retournent jamais une chaîne supérieure à 255 caractères. Si c'était le cas, le résultat retourné et inséré dans la cellule serait tronqué au-delà de cette limite.

Précisez le type du résultat

Veillez à toujours préciser le type de données renvoyé par une fonction, afin que le résultat s'affiche correctement dans la cellule. Ceci est particulièrement vrai dans le cas d'une fonction renvoyant un résultat de type Date. Excel n'appliquera en effet le format de date à la cellule que si la fonction a été déclarée comme retournant une valeur de type Date.

Des fonctions toujours à jour

Par défaut, Excel met à jour le résultat d'une fonction personnalisée dans une cellule de la même façon qu'il le fait pour les fonctions Excel. Vous pouvez cependant déclarer une fonction *volatile* de façon qu'Excel calcule ou recalcule le résultat d'une fonction chaque fois que le contenu d'une cellule de la feuille est modifié.

Pour marquer une fonction personnalisée comme volatile, insérez simplement l'instruction suivante immédiatement après l'instruction de déclaration de la fonction :

```
Application.Volatile
```

Considérez la fonction `CalculTauxBenefice` suivante. Elle retourne le taux de bénéfice réalisé à partir du chiffre d'affaires et des coûts :

```
Public Function TauxBenefice(CA As Currency, Cout As Currency) As Long
    Application.Volatile
    TauxBenefice = ((CA - Cout) / CA) * 100
End Function
```

Notre fonction reçoit les arguments de type Currency `CA` (pour Chiffre d'Affaires) et `Cout`. La première instruction de la fonction la déclare comme *volatile*. La fonction calcule ensuite le taux de bénéfice réalisé en pourcentage.

Principales fonctions VBA

Cette section présente les fonctions de calcul VBA les plus couramment utilisées dans le cadre de la programmation Excel. Elles sont classées selon les catégories suivantes :

- fonctions mathématiques ;
- fonctions de conversion de données ;
- fonctions de conversion de types de données ;
- fonctions de date et d'heure ;
- fonctions financières.

Pour obtenir une aide plus développée concernant l'utilisation de ces fonctions, consultez l'aide en ligne de VBA.



Les fonctions de traitement des chaînes de caractères constituant une question clé de la programmation, le chapitre suivant leur est entièrement consacré.



Notez que les arguments des fonctions présentées dans les tableaux suivants sont pour la plupart des arguments nommés. Lorsqu'un argument n'est pas un argument nommé, il est traduit en français.

Dans le Tableau 8.1, l'argument nommé *Number* représente une valeur numérique passée en argument.

Tableau 8.1 : Fonctions mathématiques

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
<i>Abs (Number)</i>	Retourne la valeur absolue de <i>Number</i> .	Même type que <i>Number</i>
<i>Atn (Number)</i>	Retourne l'arctangente de <i>Number</i> , sous la forme d'un angle exprimé en radians. Pour convertir des degrés en radians, multipliez-les par $\pi/180$. Inversement, pour convertir des radians en degrés, multipliez-les par $180/\pi$.	Double
<i>Cos (Number)</i>	Retourne le cosinus de l'angle <i>Number</i> (exprimé en radians).	Double

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
<i>Exp(Number)</i>	Retourne la valeur de la constante e élevée à la puissance <i>Number</i> . La constante e est la base des logarithmes népériens, et est à peu près égale à 2,718282.	Double
<i>Fix(Number)</i>	Renvoie la partie entière d'un nombre. Si <i>Number</i> est négatif, Fix renvoie le premier entier négatif supérieur ou égal à <i>Number</i> .	Même type que <i>Number</i>
<i>Int(Number)</i>	Comme la fonction Fix , la fonction Int renvoie la partie entière d'un nombre. Par contre, si <i>Number</i> est négatif, Int renvoie le premier entier négatif inférieur ou égal à <i>Number</i> .	Même type que <i>Number</i>
<i>Log(Number)</i>	Retourne le logarithme népérien de <i>Number</i> .	Double
<i>Rnd(Number)</i>	Retourne une valeur aléatoire. L'argument <i>Number</i> est facultatif et définit le comportement de la fonction Rnd . Notez que la fonction Rnd génère la même série de nombres aléatoires à chaque appel, car elle réutilise le nombre aléatoire précédent comme valeur initiale pour le calcul du nombre suivant. Utilisez l'instruction Randomize pour initialiser le générateur de nombres aléatoires à partir d'une valeur initiale tirée de l'horloge système.	Single
<i>Round(Number, NumDigitsAfterDecimal)</i>	Retourne la valeur arrondie de <i>Number</i> . L'argument facultatif <i>NumDigitsAfterDecimal</i> indique le nombre de chiffres à conserver après la virgule dans le nombre retourné. Si cet argument est omis, Round renvoie la valeur entière arrondie.	Même type que <i>Number</i>
<i>Sgn(Number)</i>	Retourne une valeur représentant le signe de <i>Number</i> : -1 si <i>Number</i> est inférieur à zéro. 0 si <i>Number</i> est égal à zéro. 1 si <i>Number</i> est supérieur à zéro.	Integer
<i>Sin(Number)</i>	Retourne le sinus de l'angle <i>Number</i> exprimé en radians.	Double
<i>Sqr(Number)</i>	Retourne la racine carrée de <i>Number</i> .	Double
<i>Tan(Number)</i>	Retourne la valeur de la tangente de l'angle <i>Number</i> exprimée en radians.	Double

Le Tableau 8.2 présente les fonctions de conversion de données de VBA. Ces fonctions traitent la valeur reçue en argument et renvoient le résultat de ce traitement.

Les fonctions de conversion de type de données sont présentées dans le Tableau 8.3.

Tableau 8.2 : Fonctions de conversion des données

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
<i>Asc(String)</i>	Renvoie une valeur représentant le code de caractères du premier caractère de la chaîne de caractères <i>String</i> .	Integer
<i>Chr(CharCode)</i>	Renvoie le caractère dont le code de caractères est <i>CharCode</i> . <i>CharCode</i> est de type Long.	Integer
<i>Format(Expression, Format)</i>	Renvoie <i>Expression</i> sous forme de chaîne formatée selon le <i>Format</i> défini. Vous pouvez ainsi définir le format d'une date, ou modifier une chaîne pour afficher le symbole de la monnaie et placer des séparateurs entre les milliers.	String
<i>Hex(Number)</i>	Retourne la valeur hexadécimale de <i>Number</i> sous forme de chaîne. Si <i>Number</i> n'est pas un nombre entier, il est arrondi à l'entier le plus proche.	String
<i>Oct(Number)</i>	Retourne la valeur octale de <i>Number</i> sous forme de chaîne.	String
<i>RGB(Red, Green, Blue)</i>	Retourne un entier représentant la valeur RGB. Les arguments <i>Red</i> , <i>Green</i> et <i>Blue</i> sont de type Integer et correspondent chacun à la valeur de la couleur associée (rouge, vert, bleu), comprise entre 0 et 255.	Long
<i>Str(Number)</i>	Renvoie <i>Number</i> sous forme de chaîne de caractères. <i>Number</i> peut être n'importe quelle valeur numérique que l'on souhaite traiter comme une chaîne.	String
<i>Val(String)</i>	Retourne <i>String</i> sous forme de valeur numérique de type approprié. Si <i>String</i> ne peut être converti en valeur numérique, une erreur est générée.	Type approprié à la valeur retournée

Contrairement aux fonctions présentées dans le tableau précédent, les fonctions décrites dans le Tableau 8.3 ne renvoient pas les données reçues en argument sous une forme différente, mais en modifiant le type. La conversion de données dans le type approprié est souvent

nécessaire au bon déroulement d'un programme VBA. Par exemple, tenter d'exécuter une fonction mathématique sur une chaîne de caractères (même si celle-ci n'est composée que de chiffres) générera une erreur. Vous devrez alors faire appel à la fonction de conversion numérique appropriée afin que l'argument passé à la fonction soit reconnu comme une valeur et non plus comme une suite de caractères.

Tableau 8.3 : Fonctions de conversion des types de données

CBool (MaVar)	Convertit <i>MaVar</i> en une valeur booléenne. Renvoie True si <i>MaVar</i> est une valeur numérique différente de 0 ; False si <i>MaVar</i> est égale à 0. Une erreur est générée si <i>MaVar</i> n'est pas une valeur numérique.
CByte (MaVar)	Convertit <i>MaVar</i> en une variable de type Byte.
CCur (MaVar)	Convertit <i>MaVar</i> en une variable de type Currency (monétaire).
CDate (MaVar)	Convertit <i>MaVar</i> en une variable de type Date.
Cdbl (MaVar)	Convertit <i>MaVar</i> en une variable de type Double.
CDec (MaVar)	Convertit <i>MaVar</i> en une variable de type Decimal.
CInt (MaVar)	Convertit <i>MaVar</i> en une variable de type Integer.
CLng (MaVar)	Convertit <i>MaVar</i> en une variable de type Long.
CSng (MaVar)	Convertit <i>MaVar</i> en une variable de type Single.
CStr (MaVar)	Convertit <i>MaVar</i> en une variable de type String. Si <i>MaVar</i> est une valeur de type Boolean, la fonction CStr renvoie Vrai ou Faux. Si <i>MaVar</i> est une valeur de type Date, la fonction CStr renvoie la date sous forme de chaîne. Si <i>MaVar</i> est une valeur de type numérique, la fonction CStr renvoie cette valeur sous forme de chaîne.
CVar (MaVar)	Convertit <i>MaVar</i> en une variable de type Variant. <i>MaVar</i> doit être une valeur de type Double pour les nombres et de type String pour les chaînes.

Le Tableau 8.4 présente les fonctions de date et d'heure. L'argument nommé *date* désigne toute expression valide qui renvoie une valeur de type Date.

Tableau 8.4 : Fonctions de date et d'heure

Fonction	Description	Type de données renvoyé
Date	Retourne la date en cours à partir de l'horloge système de votre ordinateur. Vous pouvez également utiliser l'instruction Date pour définir la date de l'horloge système.	Date

Tableau 8.4 : Fonctions de date et d'heure (*suite*)

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
Time	Retourne l'heure en cours à partir de l'horloge système. Utilisez l'instruction Time pour redéfinir l'heure système.	Date
Now	Retourne la date et l'heure en cours.	Date
Year(Date)	Retourne l'année correspondant à Date sous la forme d'un nombre entier.	Integer
Month(Date)	Retourne le mois correspondant à Date sous la forme d'un entier compris entre 1 (janvier) et 12 (décembre).	Integer
Day(Date)	Retourne le jour correspondant à Date sous la forme d'un nombre entier compris entre 1 et 31.	Integer
Weekday(Date, FirstDayOfWeek)	Retourne un entier compris entre 1 et 7, qui représente le jour de la semaine correspondant à la date Date. FirstDayOfWeek définit le jour considéré comme le premier jour de la semaine. Il peut s'agir d'une valeur comprise entre 0 et 7 ou de l'une des huit constantes vbDayOfWeek : vbUseSystem, vbMonday, vbTuesday, vbWednesday, vbThursday, vbFriday, vbSaturday et vbSunday. Si l'argument Firstdayofweek est omis, la valeur par défaut est vbUseSystem et dépend donc du système sur lequel est exécutée la macro. En France, le premier jour de la semaine par défaut est le lundi, tandis qu'aux USA il s'agira du dimanche. Il est donc recommandé de préciser l'argument Firstdayofweek afin d'éviter des différences de comportements du programme d'un ordinateur à l'autre.	Integer
WeekdayName(WeekDay, Abbreviate, FirstDayOfWeek)	Retourne une chaîne qui représente le nom du jour de la semaine WeekDay . WeekDay peut être une valeur numérique comprise entre 0 et 7, ou l'une des constantes vbDayOfWeek présentées ci-avant. La valeur retournée est renvoyée dans la langue du système sur lequel est exécutée la fonction. Sur un système en français, lundi, mardi..., dimanche.	String

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
<i>(suite)</i> WeekdayName (<i>WeekDay</i> , <i>Abbreviate</i> , <i>FirstDayOfWeek</i>)	L'argument nommé <i>Abbreviate</i> est facultatif et définit si le jour est renvoyé sous une forme abrégée (lun. pour lundi). Sa valeur par défaut est <i>False</i>. <i>FirstDayOfWeek</i> définit quel est le jour considéré comme le premier jour de la semaine. Notez que la fonction Weekday peut être utilisée comme argument de WeekdayName : ... WeekdayName (WeekDay (<i>Date</i>))	String
Hour (<i>Date</i>)	Retourne un nombre entier compris entre 0 et 23 et qui représente les heures de l'heure fournie dans <i>Date</i> . Si <i>Date</i> ne fournit pas d'information d'heure, la fonction renvoie 0.	Integer
Minute (<i>Date</i>)	Retourne un entier compris entre 0 et 59 et qui représente les minutes de l'heure fournie dans <i>Date</i> . Si <i>Date</i> ne fournit pas d'information de minutes, la fonction renvoie 0.	Integer
Second (<i>Date</i>)	Retourne un entier compris entre 0 et 59 et qui représente les secondes de l'heure fournie dans <i>Date</i> . Si <i>Date</i> ne fournit pas d'information de secondes, la fonction renvoie 0.	Integer
DateSerial (<i>Year</i> , <i>Month</i> , <i>Day</i>)	Retourne une date précise (jour, mois et année), fonction des arguments <i>Year</i> , <i>Month</i> et <i>Day</i> reçus.	Date
TimeSerial (<i>Hour</i> , <i>Minute</i> , <i>Second</i>)	Retourne une heure précise (heures, minutes et secondes), en fonction des arguments reçus.	Date
DateValue (<i>Date</i>)	Renvoie la date reçue en argument. L'argument <i>Date</i> peut être une chaîne de caractères, un nombre, une constante ou n'importe quelle expression renvoyant une date.	Date
TimeValue (<i>Date</i>)	Renvoie l'heure reçue en argument. L'argument <i>Date</i> peut être une chaîne de caractères, un nombre, une constante ou n'importe quelle expression renvoyant une heure.	Date
Timer	Renvoie le nombre de secondes écoulées depuis minuit (d'après l'horloge système de votre ordinateur).	Single

Le Tableau 8.5 présente les fonctions financières de VBA.

Tableau 8.5 : Fonctions financières

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
Fonctions de calcul d'amortissement		
DDB (<i>Cost</i> , <i>Salvage</i> , <i>Life</i> , <i>Period</i> ,)	Retourne la valeur de l'amortissement d'un bien au cours d'une période définie, en utilisant la méthode d'amortissement dégressif à taux double ou toute autre méthode précisée. Les arguments nommés reçus sont : – <i>Cost</i> : coût initial du bien. – <i>Salvage</i> : valeur du bien à la fin de son cycle de vie. – <i>Life</i> : durée du cycle de vie du bien. – <i>Period</i> : période sur laquelle l'amortissement du bien est calculé. – <i>Factor</i> : facultatif. Taux utilisé pour le calcul de l'amortissement. Si <i>Factor</i> est omis, la valeur 2 est employée (méthode d'amortissement dégressif à taux double).	Double
SLN (<i>Cost</i> , <i>Salvage</i> , <i>Life</i>)	Retourne l'amortissement linéaire d'un bien sur une période définie. Les arguments nommés sont les mêmes que pour la fonction DDB ().	Double
SYD (<i>Cost</i> , <i>Salvage</i> , <i>Life</i> , <i>Period</i>)	Retourne la valeur de l'amortissement global d'un bien sur une période donnée. Les arguments nommés sont les mêmes que pour la fonction DDB ().	Double
Fonctions de calcul de valeur future		
FV (<i>Rate</i> , <i>NPer</i> , <i>Pmt</i> , <i>PV</i> , <i>Type</i>)	Retourne le futur montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés reçus sont : – <i>Rate</i> : taux d'intérêt par échéance. Pour un prêt dont le taux annuel serait 5 %, avec des échéances mensuelles, le taux par échéance serait de 0,05/12, soit 0,0042.	Double

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
<i>(suite)</i> FV (<i>Rate</i> , <i>NPer</i> , <i>Pmt</i> , <i>PV</i> , <i>Type</i>)	– <i>NPer</i> : nombre d'échéances de l'emprunt ou du placement. – <i>Pmt</i> : montant du paiement à effectuer à chaque échéance. – <i>PV</i> : facultatif. Valeur actuelle (ou somme globale) d'une série de paiements devant être effectués dans le futur. La valeur par défaut est 0. – <i>Type</i> : facultatif. Date d'échéance des paiements. Indiquez la valeur 0 si les paiements sont dus à terme échu, ou la valeur 1 si les paiements sont dus à terme à échoir. Si l'argument est omis, la valeur par défaut est 0.	Double
Fonctions de calcul de taux d'intérêt		
Rate (<i>NPer</i> , <i>Pmt</i> , <i>PV</i> , <i>FV</i> , <i>Type</i> , <i>Guess</i>)	Retourne le taux d'intérêt par échéance pour une annuité. Les arguments nommés <i>NPer</i>, <i>Pmt</i>, <i>PV</i> et <i>Type</i> sont les mêmes que pour la fonction FV() décrite ci-dessus. – <i>FV</i> : facultatif. Valeur future ou solde que vous souhaitez obtenir après avoir effectué le dernier paiement. Si vous souhaitez épargner 10 000 \$, la valeur future est de 10 000. Si l'argument est omis, la valeur par défaut est 0 (qui est la valeur future d'un emprunt). – <i>Guess</i> : facultatif. Valeur qui devrait être renvoyée par la fonction Rate. S'il est omis, <i>Guess</i> prend la valeur 0,1 (10 pour cent).	Double
Fonctions de calcul de taux de rendement interne		
IRR (<i>Values</i> ()), <i>Guess</i>	Retourne le taux de rendement interne d'une série de mouvements de trésorerie périodiques (paiements et encaissements). Les arguments nommés sont les suivants : – <i>Values</i>() : tableau de données indiquant les valeurs des mouvements de trésorerie. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement). – <i>Guess</i> : facultatif. Valeur qui devrait être renvoyée par la fonction IRR. S'il est omis, l'argument <i>Guess</i> prend pour valeur 0,1 (10 pour cent).	Double

Tableau 8.5 : Fonctions financières (suite)

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
<code>MIRR(Values(), Finance_Rate, Reinvest_Rate)</code>	Retourne le taux de rendement interne modifié d'une série de mouvements de trésorerie périodiques (paiements et encaissements). – <i>Values()</i> : idem que pour la fonction <code>IRR</code> décrite ci-dessus. – <i>Finance_Rate</i> : taux d'intérêt payé pour couvrir le coût du financement. – <i>Reinvest_Rate</i> : taux d'intérêt perçu sur les gains tirés des sommes réinvesties.	Double
Fonctions de calcul de nombre d'échéances		
<code>NPer(Rate, Pmt, PV, FV, Type)</code>	Retourne le nombre d'échéances d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions <code>FV</code> et <code>Rate</code> décrites plus avant dans ce tableau.	Double
Fonctions de calcul de montant de versements		
<code>IPmt(Rate, per, NPer, PV, FV, Type)</code>	Renvoie une valeur de type Double indiquant le montant, sur une période donnée, d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions <code>FV</code> et <code>Rate</code> décrites plus avant dans ce tableau.	Double
<code>Pmt(Rate, NPer, PV, FV, Type)</code>	Retourne le montant d'une annuité basée sur des versements constants et périodiques et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions <code>FV</code> et <code>Rate</code> décrites plus avant dans ce tableau.	Double
<code>PPmt(Rate, per, NPer, PV, FV, Type)</code>	Retourne la valeur du remboursement du capital, pour une échéance donnée, d'une annuité basée sur des versements constants et périodiques, et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions <code>FV</code> et <code>Rate</code> décrites plus avant dans ce tableau.	Double

<i>Fonction</i>	<i>Description</i>	<i>Type de données renvoyé</i>
Fonctions de calcul de montant de valeur actuelle		
<i>NPV</i> (<i>Rate</i> , <i>Values</i> ())	Retourne la valeur nette actuelle d'un investissement, calculée en fonction d'une série de mouvements de trésorerie périodiques (paiements et encaissements) et d'un taux d'escompte. Les arguments nommés sont les suivants : – <i>Rate</i> : taux d'escompte sur la période, exprimé sous la forme d'un nombre décimal. – <i>Values</i> () : tableau de données indiquant les valeurs des mouvements de trésorerie. Le tableau doit contenir au moins une valeur négative (un paiement) et une valeur positive (un encaissement).	Double
<i>PV</i> (<i>Rate</i> , <i>NPer</i> , <i>Pmt</i> , <i>FV</i> , <i>Type</i>)	Retourne le montant actuel d'une annuité basée sur des échéances futures constantes et périodiques, et sur un taux d'intérêt fixe. Les arguments nommés sont les mêmes que pour les fonctions <i>FV</i> et <i>Rate</i> décrites plus avant dans ce tableau.	Double



9

Manipulation des chaînes de caractères

Au sommaire de ce chapitre

- Modifier des chaînes de caractères
- Comparer des chaînes de caractères
- Rechercher dans les chaînes de caractères

La manipulation de chaînes de caractères est un aspect clé de la programmation, quel que soit le langage de programmation. La manipulation de chaînes permet par exemple d'afficher des messages à l'attention de l'utilisateur pour l'informer ou pour interagir avec lui. Dans ce cadre, les valeurs numériques doivent souvent être traitées et formatées comme des chaînes de caractères afin d'en extraire les informations pertinentes. Par ailleurs, les informations retournées par la fonction `InputBox` ou *via* un contrôle `TextBox` sont des chaînes de caractères et devront être traitées comme telles.

Modifier des chaînes de caractères

Les chaînes de caractères peuvent être modifiées d'à peu près toutes les façons. Vous pouvez bien sûr ajouter des éléments à une chaîne de caractères en la concaténant avec d'autres chaînes de caractères, mais vous pouvez aussi extraire une partie d'une chaîne de façon à n'en conserver que l'information pertinente.

Concaténer des chaînes

Concaténer des chaînes consiste à les "accoler", c'est-à-dire à assembler plusieurs chaînes de caractères pour en créer une seule. Vous pouvez également concaténer l'ensemble des éléments d'une variable de matrice, de façon à obtenir une chaîne qui en reprenne toutes les valeurs.

Concaténer des chaînes simples

La concaténation des chaînes est réalisée à l'aide des opérateurs `&` et `+`. Il est cependant conseillé de privilégier l'opérateur `&`, qui ne fonctionne qu'avec les chaînes, et de conserver l'opérateur `+` pour les additions. Votre code sera ainsi sans ambiguïté.

Vous pouvez concaténer toute expression qui renvoie une chaîne de caractères. Il peut s'agir d'une variable de type `String` comme d'une fonction renvoyant une chaîne, ou encore d'une chaîne de caractères entourée de guillemets.



Vous ne pouvez pas concaténer une chaîne de caractères et une valeur numérique. Si vous souhaitez concaténer une chaîne avec une valeur numérique, utilisez la fonction `Str()` de façon à convertir la valeur numérique en une chaîne de caractères, dans l'instruction de concaténation.

Considérez l'exemple suivant :

```
1: Sub ConcatenerDesChaînes()  
2:   Dim MonNom As String  
3:   Dim MonBenefice As Long  
4:   Dim MonMessage As String  
5:   MonNom = ActiveWorkbook.ActiveSheet.Cells(1, 2)  
6:   MonBenefice = ActiveWorkbook.ActiveSheet.Cells(1, 3).Value  
7:   MonMessage = "La prime de " & MonNom & " est de " & Str(Int(MonBenefice /  
   20)) & " euros."  
8:   MsgBox MonMessage  
9: End Sub
```

Lignes 2 à 4, les variables sont déclarées. Notez que la variable `MonBenefice` est de type `Long` et ne peut donc être concaténée telle quelle. Lignes 5 et 6, les variables `MonNom` et `MonBenefice` reçoivent respectivement pour valeur le contenu des cellules A2 et A3.

Ligne 7 a lieu la concaténation des différentes chaînes de façon à générer le message affiché par la fonction `MsgBox` à la ligne suivante.

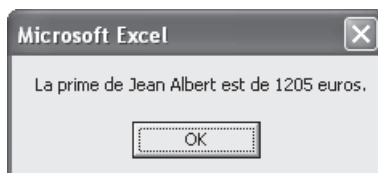
Les éléments concaténés sont les suivants :

- **"La prime de"**. Est une simple chaîne de caractères, entourée de guillemets.
- **`MonNom`**. Est une variable de type `String` et peut donc être concaténée sans qu'il soit nécessaire d'effectuer une conversion de type de données.
- **"est de"**. Est une simple chaîne de caractères. Notez que nous avons fait précéder et suivre le texte d'espaces, de manière à générer une chaîne lisible.
- **`Str(Int(MonBenefice / 20))`**. Est une expression renvoyant une chaîne de caractères. Tout d'abord, la valeur numérique `MonBenefice` est divisée par 20, de façon à obtenir la valeur de la prime (5 % du bénéfice). La fonction `Int` renvoie la valeur entière de la somme ainsi obtenue. Enfin, la fonction `Str` convertit le résultat numérique ainsi obtenu en une chaîne de caractères qui peut ainsi être concaténée avec les autres chaînes.
- **"euros"**. Est une simple chaîne de caractères.

On obtient ainsi le message affiché dans la boîte de dialogue de la Figure 9.1.

Figure 9.1

La concaténation de chaînes permet d'afficher des messages à l'attention de l'utilisateur.



Concaténer les valeurs d'une variable de matrice

Si vous souhaitez créer une seule chaîne de caractères à partir des différentes valeurs d'une variable de matrice (ou variable tableau), utilisez la fonction `Join` selon la syntaxe suivante :

```
Join(SourceArray, Delimiter)
```

où l'argument nommé *SourceArray* est la variable de matrice dont on souhaite réunir les données en une chaîne, et *Delimiter* (facultatif) la chaîne de caractères qui sera utilisée comme séparateur entre les différentes données extraites de la variable. Si *Delimiter* est omis, les éléments du tableau sont concaténés sans séparateur.

Dans l'exemple suivant, les jours de la semaine sont stockés dans le tableau `JoursSemaine`. La boîte de dialogue représentée à la Figure 9.2 est ensuite affichée. Notez qu'on utilise une virgule suivie d'un espace comme séparateur, et que l'on ajoute un point à la fin de la chaîne.

```
Sub ConcatenerUnTableau()
    Dim JoursSemaine(1 To 7)
    Dim n as Byte
    For n = 1 To 7
        JoursSemaine(n) = WeekdayName(Weekday:=n, abbreviate:=False, _
            firstdayofweek:=vbMonday)
    Next n
    MsgBox "Les jours de la semaine sont : " & Join(JoursSemaine, ", ") & "."
End Sub
```

Figure 9.2

Utilisez la fonction Join pour concaténer les espaces de stockage d'une variable tableau.



Insérer des caractères non accessibles au clavier

Pour insérer, dans des chaînes, des caractères qui ne peuvent être saisis dans le code VBA à l'aide du clavier – tels qu'un saut de paragraphe ou un retour à la ligne –, on fait appel à la fonction `Chr()` qui renvoie le caractère dont le code est précisé entre parenthèses.

Utilisez la fonction `Chr()` selon la syntaxe suivante :

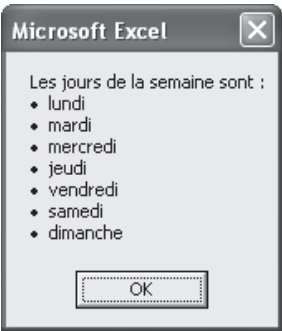
```
Chr(CharCode)
```

où l'argument nommé *CharCode* (de type Long) est le code ASCII du caractère que l'on souhaite insérer. Dans l'exemple suivant, on utilise la fonction `Chr()` pour insérer un retour

chariot (Chr (13)) et une puce (Chr (149)) devant chaque jour de la semaine, afin de générer le message ensuite affiché dans une boîte de dialogue (voir Figure 9.3).

```
Sub UtiliserChr()  
    Dim n As Byte  
    Dim Message As String  
    Message = "Les jours de la semaine sont :"  
    For n = 1 To 7  
        Message = Message & Chr(13) & Chr(149) & " " & WeekdayName(Weekday:=n,  
            abbreviate:=False, firstdayofweek:=vbMonday)  
    Next n  
    MsgBox Message  
End Sub
```

Figure 9.3
*Utilisez la fonction Chr()
pour insérer des caractères
non accessibles au clavier.*



Afin d’améliorer la lisibilité de votre code, préférez les constantes Visual Basic à la fonction Chr () quand cela est possible. Le Tableau 9.1 présente les constantes Visual Basic les plus communément employées pour créer des chaînes de caractères.

Tableau 9.1 : Constantes Visual Basic renvoyant un caractère

Constante	Valeur	Equivalent avec Chr ()
vbCr	Saut de paragraphe	Chr (13)
vbLf	Saut de ligne	Chr (10)
vbCrLf	Retour chariot + saut de ligne	Chr (13) & Chr (10)
vbNewLine	Saut de ligne spécifique à la plate- forme : équivaut à vbCrLf pour Win- dows et à vbCr sur un Macintosh	Chr (13) & Chr (10) – Windows Chr (13) – Macintosh
vbTab	Tabulation	Chr (9)

Répéter une série de caractères

Utilisez la fonction `String()` pour renvoyer une chaîne composée d'un caractère répété un nombre défini de fois, selon la syntaxe suivante :

```
String(Number, Character)
```

où les arguments nommés *Number* et *Character* représentent respectivement le nombre de répétitions et le caractère à répéter. L'expression suivante insère dans une chaîne la valeur de la cellule A1 à laquelle sont ajoutés six zéros. La valeur, exprimée en millions dans la cellule, est ainsi récupérée sous forme d'unités :

```
MsgBox Str(ActiveWorkbook.ActiveSheet.Range("A1").Value) & String(6, "0")
```



*Vous pouvez utiliser la fonction `Chr()` pour l'argument *Character*. Ainsi `String(10, Chr(149))` renvoie une chaîne composée de dix puces.*

Utilisez la fonction `Space` pour renvoyer une chaîne composée d'un nombre défini d'espaces, selon la syntaxe suivante :

```
Space(Number)
```

où l'argument nommé *Number* est le nombre d'espaces à renvoyer.

Supprimer les espaces superflus d'une chaîne

Traiter les espaces d'une chaîne est une opération souvent indispensable au bon fonctionnement d'un programme. Par exemple, si vous recevez des informations de l'utilisateur grâce à la fonction `InputBox`, il sera prudent de supprimer les éventuels espaces superflus à l'une ou l'autre des extrémités de la chaîne retournée avant de traiter ces données.

Visual Basic propose trois fonctions permettant de supprimer les espaces superflus d'une chaîne :

- **LTrim(*String*)**. Retourne une chaîne de caractères à partir de la chaîne *String* reçue en argument, dont les espaces situés à gauche ont été supprimés.
- **RTrim(*String*)**. Retourne une chaîne de caractères à partir de la chaîne *String* reçue en argument, dont les espaces situés à droite ont été supprimés.
- **Trim(*String*)**. Retourne une chaîne de caractères à partir de la chaîne *String* reçue en argument, dont les espaces situés à droite et à gauche ont été supprimés.

L'argument nommé *String* est une chaîne de caractères.

L'instruction suivante stocke les données saisies par l'utilisateur dans la variable `MaVar` après avoir pris soin de supprimer les éventuels espaces saisis par erreur :

```
MaVar = Trim(InputBox("Veuillez saisir votre nom :"))
```

Extraire une partie d'une chaîne

Les fonctions `Left()`, `Right()` et `Mid()` permettent de récupérer une partie d'une chaîne de caractères. Combinées aux fonctions de recherche dans une chaîne (présentées plus loin dans ce chapitre), elles permettent d'extraire des informations précises d'une chaîne. La fonction `Len()` – qui renvoie la longueur d'une chaîne, c'est-à-dire le nombre de caractères qui la composent – est souvent employée comme argument de ces fonctions. Utilisez ces fonctions selon la syntaxe suivante :

- **`Len(String)`**. Retourne une valeur de type `Double` qui représente le nombre de caractères présents dans la chaîne `String`.
- **`Left(String, Lenght)`**. Retourne les *Lenght* premiers caractères situés à gauche de la chaîne *String* reçue en argument. Si *Lenght* est supérieur ou égal à la longueur de la chaîne *String*, la totalité de la chaîne est retournée.
- **`Right(String, Lenght)`**. Retourne les *Lenght* premiers caractères situés à droite de la chaîne *String* reçue en argument. Si *Lenght* est supérieur ou égal à la longueur de la chaîne *String*, la totalité de la chaîne est retournée.
- **`Mid(String, Start, Length)`**. Retourne une chaîne composée des *Lenght* caractères situés dans la chaîne *String* à partir du caractère dont la position est *Start*. Le caractère situé à la position *Start* est également retourné. L'argument *Length* est facultatif. S'il est omis ou s'il est supérieur au nombre de caractères présents après la position *Start*, tous les caractères situés à partir de la position *Start* et jusqu'à la fin de la chaîne sont retournés.

Considérez les trois instructions suivantes :

```
1: MonTexte = Left(MonTexte, Len(MonTexte) - 1)
2: MonTexte = Right(MonTexte, Len(MonTexte) - 1)
3: Trim(Mid(ActiveWorkbook.ActiveSheet.Range("A1").Value, InStr(ActiveWorkbook.
ActiveSheet.Range("A1").Value, Chr(34)) + 1))
```

La première instruction modifie la variable `MonTexte` en supprimant le dernier caractère (à droite). On utilise pour cela la fonction `Len()` qui retourne la longueur de la chaîne à laquelle on retire 1.

Ligne 2, on modifie la variable `MonTexte` en supprimant le premier caractère (à gauche). Comme dans l'instruction précédente, on utilise la fonction `Len()` pour retourner la longueur de la chaîne à laquelle on enlève 1.

Enfin, l'instruction de la ligne 3 retourne le contenu de la cellule A1 à partir du premier guillemet (non inclus) présent dans la chaîne. On utilise pour cela la fonction `InStr()` présentée plus loin dans ce chapitre, qui retourne la position du caractère recherché (ici `Chr(34)`, soit un guillemet). On utilise cette valeur à laquelle on ajoute 1 pour commencer la chaîne à partir du caractère qui suit le guillemet. Notez que l'on emploie la fonction `Trim()` afin de supprimer les éventuels espaces présents aux extrémités de la chaîne retournée par la fonction `Mid()`.

Effectuer des remplacements au sein d'une chaîne

Utilisez la fonction `Replace()` pour effectuer des remplacements au sein d'une chaîne de caractères selon la syntaxe suivante :

`Replace(Expression, Find, Replace, Start, Count, Compare)`

où *Expression* est une expression valide qui renvoie la chaîne de caractères à traiter. *Find* est la chaîne recherchée et est remplacée par *Replace*. Les trois arguments suivants sont facultatifs :

- **Start.** Précise la position de caractères à partir de laquelle doivent s'effectuer les remplacements dans *Expression*. Si cet argument est omis, l'ensemble de la chaîne est traité.
- **Count.** Indique le nombre de remplacements à effectuer. S'il est omis, sa valeur par défaut – 1 est utilisée, et toutes les occurrences de *Find* sont remplacées.

Enfin, *Compare* précise le type de comparaison effectuée. Il peut s'agir de l'une des valeurs suivantes :

- **vbUseCompareOption ou -1.** C'est la valeur de l'option `Option Compare` précisée dans l'en-tête du module dans lequel se trouve l'instruction qui est utilisée pour définir le type de comparaison.
- **vbBinaryCompare ou 0.** Effectue une comparaison binaire. La recherche se fait en respectant la casse de *Find*.
- **vbTextCompare ou 1.** Effectue une comparaison de texte. La recherche se fait sur le texte, sans prendre en considération la casse des occurrences trouvées.
- **vbDatabaseCompare ou 2.** Effectue une comparaison s'appuyant sur des informations contenues dans une base de données. Microsoft Access seulement.

Dans l'exemple suivant, lignes 7 à 9, les données de la feuille active sont stockées dans la variable de matrice `MesValeurs`, tout en remplaçant au passage les ";" par des ",".

```
1: Sub UtiliserReplace()  
2:   Dim MaCellule As Range  
3:   Dim n As Long
```

```
4: Dim MesValeurs()  
5: ReDim MesValeurs(1 To ActiveWorkbook.ActiveSheet.UsedRange.Cells.Count)  
6: n = 0  
  
7: For Each MaCellule In ActiveWorkbook.ActiveSheet.UsedRange.Cells  
8:     n = n + 1  
9:     MesValeurs(n) = Replace(MaCellule.Value, ";", ",")  
10: Next MaCellule  
11: End Sub
```

Astuce

Affectez une chaîne nulle à l'argument `Replace` pour supprimer toutes les occurrences de `Find` dans la chaîne traitée.

Modifier la casse des chaînes de caractères

Les fonctions suivantes permettent de formater une chaîne :

- **StrConv(String, Conversion)**. Convertit les majuscules et minuscules de la chaîne *String* selon la valeur de *Conversion* :
 - vbUpperCase ou 1 convertit la chaîne en majuscules.
 - vbLowerCase ou 2 convertit la chaîne en minuscules.
 - vbProperCase ou 3 convertit les lettres de la chaîne en minuscules et applique une majuscule à la première lettre de chaque mot.
- **UCase(String)**. Retourne la chaîne *String* en convertissant toutes les lettres minuscules en majuscules. Équivaut à utiliser la fonction StrConv() en affectant la valeur 1 ou vbUpperCase à l'argument *Conversion*.
- **LCase(String)**. Retourne la chaîne *String* en convertissant toutes les lettres majuscules en minuscules. Équivaut à utiliser la fonction StrConv() en affectant la valeur 2 ou vbLowerCase à l'argument *Conversion*.

Comparer des chaînes de caractères

Comme vous le verrez en développant vos programmes VBA, la comparaison de chaînes de caractères est souvent utilisée dans des structures de contrôle pour répéter une opération jusqu'à ce qu'une condition soit remplie.

La méthode la plus simple pour comparer des chaînes consiste à employer les opérateurs arithmétiques =, < et >. Vous pouvez également utiliser la fonction `StrComp()`, en respectant la syntaxe suivante :

`StrComp(String1, String2, Compare)`

où *String1* et *String2* sont les chaînes à comparer. L'argument facultatif *Compare* définit le type de comparaison à effectuer et peut prendre les valeurs suivantes :

- **vbUseCompareOption** ou **-1**. C'est la valeur de l'option `Option Compare` précisée dans l'en-tête du module dans lequel se trouve l'instruction qui est utilisée pour définir le type de comparaison.
- **vbBinaryCompare** ou **0**. Effectue une comparaison binaire. La recherche se fait en respectant la casse de *Find*.
- **vbTextCompare** ou **1**. Effectue une comparaison de texte. La recherche se fait sur le texte, sans prendre en considération la casse des occurrences trouvées.
- **vbDatabaseCompare** ou **2**. Effectue une comparaison s'appuyant sur des informations contenues dans une base de données. Microsoft Access seulement.

La fonction `StrComp()` peut renvoyer les valeurs suivantes :

- **-1**. *String1* est inférieur à *String2*.
- **0**. *String1* est égal à *String2*.
- **1**. *String1* est supérieur à *String2*.
- **Null**. *String1* ou *String2* est de type Null.

Pour comparer des chaînes de caractères, la fonction `StrComp()` compare les codes ANSI des caractères qui les composent par ordre de position. Lorsque le code de deux caractères comparés est différent, la comparaison s'arrête et la fonction renvoie la valeur correspondante. Si les deux caractères comparés sont identiques, les caractères suivants des deux chaînes sont à leur tour comparés, etc.

Le tableau suivant présente quelques exemples de comparaisons de chaînes en mode de comparaison `vbTextCompare` :

<i>String1</i>	<i>String2</i>	<i>Résultat de la comparaison</i>
a	A	Les deux chaînes sont identiques.
abc	AbC	Les deux chaînes sont identiques.
ab	ba	<i>String1</i> est inférieur à <i>String2</i> .
abc	a	<i>String1</i> est supérieur à <i>String2</i> .

La procédure suivante fait appel à la fonction *InputBox()* pour afficher à deux reprises une boîte de saisie dans laquelle l'utilisateur est invité à entrer une chaîne de caractères. Les deux chaînes sont ensuite comparées et le résultat de la comparaison s'affiche dans une boîte de dialogue grâce à l'instruction *MsgBox*.

```
1: Sub ComparerDesChaînes()  
2:   Dim MaChaîne1 As String  
3:   Dim MaChaîne2 As String  
4:   MaChaîne1 = InputBox("Veuillez entrer la valeur de la chaîne 1 :",  
     "Comparaison de chaînes")  
5:   Do Until MaChaîne1 <> ""  
6:     MaChaîne1 = InputBox("Vous devez préciser une chaîne à comparer !" & _  
7:       vbCrLf & "Entrez une valeur pour la chaîne 1 :", "Comparaison de chaînes")  
8:   Loop  
9:   MaChaîne2 = InputBox("Veuillez entrer la valeur de la chaîne 2 :",  
     "Comparaison de chaînes")  
10:  Do Until MaChaîne2 <> ""  
11:    MaChaîne2 = InputBox("Vous devez préciser une chaîne à comparer !" & _  
12:      vbCrLf & "Entrez une valeur pour la chaîne 2 :", "Comparaison de chaînes")  
13:  Loop  
14:  Select Case StrComp(MaChaîne1, MaChaîne2, vbTextCompare)  
15:    Case -1  
16:      MsgBox "La chaîne 1 est inférieure à la chaîne 2."  
17:    Case 0  
18:      MsgBox "La chaîne 1 et la chaîne 2 sont identiques."  
19:    Case 1  
20:      MsgBox "La chaîne 1 est supérieure à la chaîne 2."  
21:  End Select  
22: End Sub
```

Lignes 5 à 8 pour la première chaîne et 10 à 13 pour la seconde, une structure *Do Until... Loop* est employée pour afficher de nouveau une boîte de saisie si l'utilisateur a validé avec une zone de saisie vide. On utilise ici l'opérateur *<>* pour comparer deux chaînes (la variable et une chaîne vide) afin de définir si la boucle doit être exécutée.

Lignes 14 à 21, une structure *Select Case...End Select* est utilisée pour définir la valeur renvoyée par l'instruction *StrComp(MaChaîne1, MaChaîne2, vbTextCompare)* et afficher le message adéquat.

Rechercher dans les chaînes de caractères

La recherche de chaînes dans des chaînes de caractères est l'une des opérations les plus courantes du traitement de chaînes. VBA permet de rechercher une chaîne de caractères au sein d'une autre chaîne, mais aussi de rechercher des chaînes de caractères dans des variables de matrice.

Rechercher une chaîne dans une chaîne

Pour rechercher une chaîne au sein d'une autre chaîne, faites appel à la fonction `InStr()`. Elle renvoie la position dans une chaîne de la première occurrence de la chaîne qui y est recherchée. `InStr()` est utilisée selon la syntaxe suivante :

```
InStr(start, string1, string2, compare)
```

où *string1* et *string2* sont respectivement la chaîne dans laquelle s'effectue la recherche et la chaîne à rechercher. L'argument *start* est facultatif et indique la position du caractère à partir duquel la chaîne doit être recherchée. S'il est omis, la recherche commence dès le premier caractère. L'argument facultatif *compare* définit le type de comparaison à effectuer. Il peut prendre l'une des valeurs suivantes :

- **vbUseCompareOption ou -1.** C'est la valeur de l'option `Option Compare` précisée dans l'en-tête du module dans lequel se trouve l'instruction qui est utilisée pour définir le type de comparaison.
- **vbBinaryCompare ou 0.** Effectue une comparaison binaire. La recherche se fait en respectant la casse de *Find*.
- **vbTextCompare ou 1.** Effectue une comparaison de texte. La recherche se fait sur le texte, sans prendre en considération la casse des occurrences trouvées.
- **vbDatabaseCompare ou 2.** Effectue une comparaison s'appuyant sur des informations contenues dans une base de données, Microsoft Access uniquement.

Dans l'exemple suivant la procédure `MaMacro` appelle la fonction `VerifierEnregistrement` pour s'assurer que le document actif est enregistré avant de s'exécuter. Si ce n'est pas le cas, la fonction renvoie `False`, et l'instruction `End` met fin à l'exécution du programme après avoir affiché un message à l'attention de l'utilisateur.

```
1: Sub MaMacro()  
2:   If VerifierEnregistrement(ActiveDocument) = False Then  
3:     MsgBox "Cette commande ne peut être exécutée que sur un fichier  
       enregistré.", _  
4:     vbOKOnly + vbCritical, "Exécution impossible"  
5:   End
```

```

6:      End If
7:      'suite des instructions de la macro
8: End Sub

9: Function VerifierEnregistrement(MonDoc As Document) As Boolean
10:     If InStr(MonDoc.FullName, Application.PathSeparator) = 0 Then
11:         VerifierEnregistrement = False
12:     Else
13:         VerifierEnregistrement = True
14:     End If
15: End Function

```

La fonction `VerifierEnregistrement` est appelée ligne 2 et reçoit en argument le document actif. Ligne 10, on recherche le séparateur propre au système sur lequel s'exécute le programme (`Application.PathSeparator`) au sein du nom complet du document, chemin inclus (propriété `FullName`). Si le fichier n'a pas été enregistré, la chaîne retournée par la propriété `FullName` est la même que celle qui serait retournée par la propriété `Name` et ne contient pas de séparateur ; la fonction `InStr()` renvoie donc la valeur 0 et notre fonction reçoit la valeur `False`. Dans le cas contraire, c'est la valeur `True` qui est affectée à la fonction `VerifierEnregistrement`.

Dans l'exemple suivant, la fonction `ScinderChaine` est appelée par `MaMacro` et reçoit en arguments deux chaînes de caractères : la chaîne à scinder en plusieurs parties et le séparateur à utiliser pour effectuer cette opération. Elle renvoie ensuite une variable de matrice dont les zones de stockage contiennent les différentes chaînes extraites de la chaîne reçue en argument. Celles-ci sont ensuite traitées de façon à être affichées.

```

1: Sub MaMacro()
2:     Dim n As Long
3:     Dim MonResultat()
4:     Dim Exemple As String
5:     Dim Message As String
6:     Exemple = "Transports:Maritimes:Bateaux:A voile"
7:     MonResultat = ScinderChaine(Exemple, ":")
8:     For n = 1 To UBound(MonResultat)
9:         Message = Message & vbCrLf & Chr(149) & _
            " " & MonResultat(n)
10:    Next n
11:    MsgBox "La chaîne a été scindée en " & UBound(MonResultat) & _
        " parties :" & Message
12: End Sub

13: Function ScinderChaine (machaine As String, separateur As String)
14:     Dim pos As Long

```

```

15:    Dim mavar
16:    ReDim mavar(1)
17:    pos = InStr(machaine, separateur)
18:    Do While pos <> 0
19:        mavar(UBound(mavar)) = Left(machaine, pos - 1)
20:        machaine = Mid(machaine, pos + Len(separateur))
21:        pos = InStr(machaine, separateur)
22:        ReDim Preserve mavar(UBound(mavar) + 1)
23:    Loop
24:    mavar(UBound(mavar)) = machaine
25:    ScinderChaine = mavar
26: End Function

```

Ligne 7, la fonction `ScinderChaine` est appelée et reçoit en arguments la chaîne affectée à la variable `Exemple` à la ligne précédente, ainsi que le séparateur à utiliser, en l'occurrence deux-points.

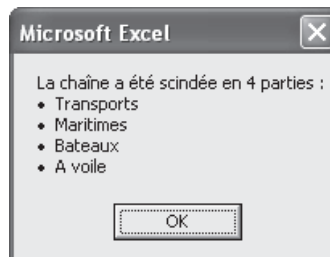
La fonction `ScinderChaine` prend alors la main. La variable de matrice `mavar` est déclarée puis initialisée à la ligne suivante. Ligne 17, `pos` reçoit en valeur le résultat de la recherche du séparateur au sein de la chaîne.

Lignes 18 à 23, une structure de contrôle `Do While...Loop` est employée afin de répéter la recherche tant qu'elle aboutit, c'est-à-dire tant que la fonction `InStr()` renvoie une valeur différente de 0. À chaque nouvelle recherche, on affecte au dernier espace de stockage de `mavar` la chaîne précédant la position du séparateur trouvé (`Left(machaine, pos - 1)`). La valeur de `machaine` est ensuite redéfinie de façon à en supprimer la partie extraite, ainsi que le séparateur recherché (ligne 20). On utilise pour cela les fonctions `Mid()` et `Len()` afin de définir la position de départ de la nouvelle chaîne immédiatement après le séparateur (`pos + Len(separateur)`). La comparaison est alors de nouveau effectuée sur la nouvelle chaîne (ligne 21) et la variable de `mavar` reçoit un espace de stockage supplémentaire. Notez l'utilisation du mot clé `Preserve` afin de préserver les valeurs déjà stockées dans `mavar`.

Enfin, ligne 24, le dernier espace de stockage de `mavar` reçoit la valeur de `machaine` quand la recherche n'a pas abouti. La procédure appelante reprend alors la main et le message représenté à la Figure 9.4 s'affiche.

Figure 9.4

Utilisez la fonction `InStr()` pour scinder une chaîne fonction d'un séparateur.



Rechercher une chaîne dans une variable de matrice

Visual Basic propose deux fonctions pour rechercher une chaîne au sein de tous les espaces de stockage d'une variable de matrice : `Filter()` et `Split()`.

La fonction `Filter()` renvoie un tableau de base zéro dont les espaces de stockage sont les mêmes que ceux de la variable traitée, dans lesquels la recherche a abouti. Si la recherche n'a pas réussi, la fonction renvoie `-1`. Utilisez la fonction `Filter()` en respectant la syntaxe suivante :

```
Filter(sourcearray, match, include, compare)
```

où *sourcearray* représente la variable de matrice à une dimension dans laquelle s'effectue la recherche, et *match* la chaîne recherchée. *include* et *compare* sont facultatifs. Si *include* est défini à `True` (valeur par défaut), les contenus des zones de stockage où a abouti la recherche sont retournés. Si *include* est défini à `False`, ce sont les zones de stockage dans lesquelles n'a pas abouti la recherche qui sont renvoyées.

Dans l'exemple suivant, on affecte les noms des jours de la semaine aux différents espaces de stockage d'une variable de matrice. On effectue ensuite quatre recherches distinctes au sein de la variable et on en affiche les résultats.

```
1: Sub RechercherDansUnTableau()  
2:   Dim n As Long  
3:   Dim JoursSemaine(1 To 7)  
4:   Dim MaRecherche1  
5:   Dim MaRecherche2  
6:   Dim MaRecherche3  
7:   Dim MaRecherche4  
8:   Dim Message As String  
9:   For n = 1 To 7  
10:      JoursSemaine(n) = WeekdayName(Weekday:=n, abbreviate:=False,  
          firstdayofweek:=vbMonday)  
11:   Next n  
12:   MaRecherche1 = Filter(sourcearray:=JoursSemaine(), match:="M",  
       include:=True, Compare:=vbBinaryCompare)  
13:   MaRecherche2 = Filter(sourcearray:=JoursSemaine(), match:="M",  
       include:=False, Compare:=vbBinaryCompare)  
14:   MaRecherche3 = Filter(sourcearray:=JoursSemaine(), match:="M",  
       include:=True, Compare:=vbTextCompare)  
15:   MaRecherche4 = Filter(sourcearray:=JoursSemaine(), match:="M",  
       include:=False, Compare:=vbTextCompare)  
16:   If Not UBound(MaRecherche1) = -1 Then  
17:      Message = "- Résultat de la recherche avec include:=True et  
          Compare:=vbBinaryCompare : " & vbCrLf
```



```
18:         For n = 0 To UBound(MaRecherche1)
19:             Message = Message & " " & MaRecherche1(n) & " / "
20:         Next n
21:     Else
22:         Message = "- Pas de résultat pour la recherche avec include:=True et
Compare:=vbBinaryCompare."
23:     End If
24:     If Not UBound(MaRecherche2) = -1 Then
25:         Message = Message & vbCr & "- Résultat de la recherche avec
include:=False et Compare:=vbBinaryCompare " & vbCr
26:         For n = 0 To UBound(MaRecherche2)
27:             Message = Message & " " & MaRecherche2(n) & " / "
28:         Next n
29:     Else
30:         Message = Message & vbCr & "- Pas de résultat pour la recherche avec
include:=False et Compare:=vbBinaryCompare."
31:     End If
32:     If Not UBound(MaRecherche3) = -1 Then
33:         Message = Message & vbCr & "- Résultat de la recherche avec
include:=True et Compare:=vbTextCompare" & vbCr
34:         For n = 0 To UBound(MaRecherche3)
35:             Message = Message & " " & MaRecherche3(n) & " / "
36:         Next n
37:     Else
38:         Message = Message & vbCr & "- Pas de résultat pour la recherche avec
include:=True et Compare:=vbTextCompare."
39:     End If
40:     If Not UBound(MaRecherche4) = -1 Then
41:         Message = Message & vbCr & "- Résultat de la recherche avec
include:=False et Compare:=vbTextCompare : " & vbCr
42:         For n = 0 To UBound(MaRecherche4)
43:             Message = Message & " " & MaRecherche4(n) & " / "
44:         Next n
45:     Else
46:         Message = Message & vbCr & "- Pas de résultat pour la recherche avec
include:=False et Compare:=vbTextCompare."
47:     End If
48:     MsgBox Message
49: End Sub
```

Lignes 9 à 11, le nom des jours de la semaine est affecté aux sept espaces de stockage de la variable de matrice JoursSemaine. On fait pour cela appel à la fonction WeekdayName() présentée au chapitre précédent. Lignes 12 à 15, on recherche à quatre reprises la chaîne

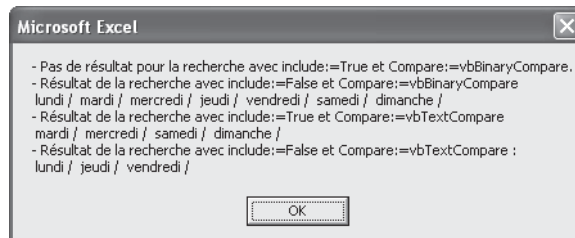
"M" dans la variable avec, chaque fois, des valeurs différentes pour les arguments *include* et *compare*.

Les mêmes instructions sont ensuite appliquées aux quatre variables ayant reçu pour valeur les résultats des recherches, afin de créer le message qui sera affiché (lignes 16 à 23, 24 à 31, 32 à 39 et 40 à 47). On utilise pour cela une structure *If...Then...Else* afin de savoir si la recherche a abouti. Si c'est le cas, une boucle *For...Next* est utilisée pour concaténer le contenu des espaces de stockage de la variable tableau ayant reçu le résultat de la fonction.

Enfin, on affiche le message représenté à la Figure 9.5.

Figure 9.5

La fonction Filter() permet d'effectuer des recherches inclusives ou exclusives.



Considérez les résultats de la recherche :

- Dans le premier cas, la recherche ne renvoie aucun résultat, car l'argument *Compare* est défini à *vbBinaryCompare* et s'effectue donc en respectant la casse (M majuscule et non minuscule).
- L'argument *Compare* de la deuxième recherche est également défini à *vbBinaryCompare*. Par contre, la fonction renvoie toutes les valeurs de la variable *JoursSemaine* puisque cette fois-ci l'argument *include* a été défini à *False* (tous les espaces de stockage où la recherche n'a pas abouti sont renvoyés).
- Pour la troisième recherche, les arguments *include* et *compare* ont respectivement été définis à *True* et *vbTextCompare*. La recherche s'applique donc sans respecter la casse et toutes les zones de stockage contenant la lettre m (minuscule ou majuscule) sont retournées.
- Enfin, la quatrième recherche est effectuée en mode *vbTextCompare*, mais cette fois-ci l'argument *include* a été défini à *False*. Ce sont donc tous les espaces de stockage ne contenant pas la lettre M (majuscule) ou m (minuscule) qui sont retournés.



10

Débogage et gestion des erreurs

Au sommaire de ce chapitre

- Les étapes et les outils du débogage
- Exemple de débogage
- Gestion des erreurs et des exceptions
- Exemple de gestion d'erreur

Il arrivera inmanquablement que des erreurs surviennent lors de l'exécution d'un programme VBA ou que le résultat d'un programme ne soit pas celui qui était escompté. Vous devrez alors déterminer l'origine de l'erreur et tester de nouveau le programme. VBA dispose pour cela de précieux outils. Ce chapitre vous les présente.

Mais vous devez tout d'abord distinguer le *débogage* de la *gestion des erreurs*. Le débogage consiste à corriger le code lorsqu'un programme ne fonctionne pas à cause d'un problème lié à une erreur dans le code : faute de frappe, syntaxe incorrecte, etc. La *gestion des erreurs* consiste à prévoir les éventuelles erreurs susceptibles de survenir et à y remédier. Un programme peut avoir été testé avec succès au moment de sa création et générer des erreurs lors de son exécution dans un contexte différent. La gestion des erreurs consiste à rendre un programme aussi fiable que possible, c'est-à-dire s'exécutant correctement dans des contextes différents.

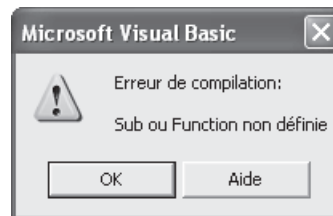
Les étapes et les outils du débogage

Le débogage consiste donc à régler les erreurs directement liées au code du programme et indépendantes de l'environnement dans lequel s'exécute le programme. Trois types d'erreurs peuvent affecter un programme VBA :

- **Erreurs de compilation.** Ce type d'erreur survient lorsque VBA rencontre une instruction qu'il ne reconnaît pas ; par exemple, lorsqu'un mot clé contient une faute d'orthographe (voir Figure 10.1).

Figure 10.1

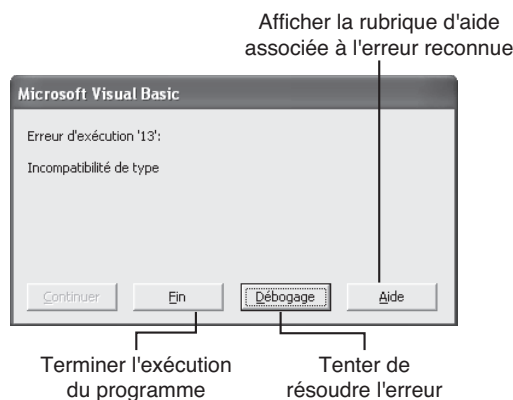
Les erreurs de compilation sont les plus faciles à repérer.



- **Erreurs d'exécution.** Une erreur d'exécution survient après que la compilation du programme a été réalisée avec succès. Une erreur d'exécution peut, par exemple, être liée à l'utilisation de données incompatibles. Ce sera le cas si le programme effectue une opération arithmétique sur une variable de chaîne (voir Figure 10.2).

Figure 10.2

Une valeur et un message définissent le type d'erreur reconnue par Visual Basic.



- **Erreurs logiques.** Les erreurs logiques sont les plus difficiles à redresser. Contrairement aux erreurs de compilation et d'exécution, elles laissent le programme s'exécuter. Le résultat obtenu ne sera pas celui que vous escomptiez. Les sections suivantes présentent les outils de débogage qui permettront de détecter l'origine de l'erreur.



L'activation de l'option Vérification automatique de la syntaxe (voir Chapitre 5) de Visual Basic Editor permet de repérer les erreurs de syntaxe dès l'étape de la saisie du code.



Il est recommandé de forcer la déclaration explicite des variables à l'aide de l'instruction `Option Explicit`. Vous éviterez ainsi tout risque d'erreur lié à une faute de frappe lors de la saisie d'un nom de variable. Pour plus d'informations reportez-vous au Chapitre 6.

Test du projet

Il est d'usage de tester tout nouveau programme lors du développement. Le premier test consiste à compiler le projet, afin de détecter les éventuelles erreurs d'exécution. Lors de la compilation, Visual Basic Editor teste chaque instruction du projet. Pour compiler un projet, choisissez la commande Compiler du menu Débogage. Si une erreur est mise en évidence, remédiez-y. Et recommencez jusqu'à ce que la compilation se déroule normalement.



Lorsque la compilation s'effectue correctement, testez le projet à partir de Visual Basic Editor : cliquez sur le bouton Exécuter de la barre d'outils Standard.



Ne testez pas un programme dont vous n'êtes pas certain sur un document sensible. Il est préférable d'effectuer les tests sur une copie. Vous serez ainsi assuré de ne pas endommager des données précieuses si le programme ne s'exécute pas correctement, ou s'il produit des données erronées.



Si une erreur d'exécution est générée, l'instruction coupable est mise en évidence. Remédiez-y. Pour renouveler la procédure, réinitialisez le projet. Cliquez pour cela sur le bouton Réinitialiser de la barre d'outils Standard ou choisissez la commande Réinitialiser du menu Exécution. La mise en évidence de l'instruction source de l'erreur d'exécution disparaît. Exécutez à nouveau la procédure, et ce jusqu'à ce que le programme s'exécute normalement.

Lorsque survient une erreur de compilation ou d'exécution, il est aisé de repérer l'instruction erronée. Si le projet a été compilé à partir de Visual Basic Editor, l'instruction incriminée est mise en évidence. Si le programme est exécuté à partir de l'application hôte, la boîte de dialogue de la Figure 10.2 apparaît. Un clic sur le bouton Débogage ouvre Visual Basic Editor (si nécessaire) et affiche la procédure dans laquelle se trouve l'instruction ayant généré l'erreur. Par défaut, celle-ci apparaît en jaune et est signalée par un indicateur de marge (voir Figure 10.3).

Figure 10.3

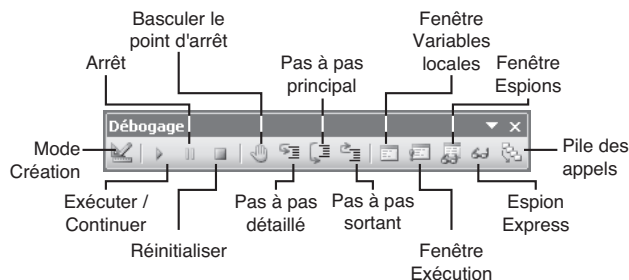
L'instruction ayant généré l'erreur est mise en évidence dans Visual Basic Editor.

```
Function ScinderChaine(machaine As String, separateur As String)
    Dim pos As Long
    Dim mavar
    ReDim mavar(1)
    pos = InStr(machaine, separateur, 10)
    Do While pos <> 0
        mavar(UBound(mavar)) = Left(machaine, pos - 1)
        machaine = Mid(machaine, pos + Len(separateur))
        pos = InStr(machaine, separateur)
        ReDim Preserve mavar(UBound(mavar) + 1)
    Loop
    mavar(UBound(mavar)) = machaine
    ScinderChaine = mavar
End Function
```

Il est cependant plus difficile de repérer les erreurs logiques. Les outils de débogage de Visual Basic Editor sont alors d'un grand secours. Ils sont accessibles *via* le menu Débogage et, pour les plus usités, *via* la barre d'outils Débogage.

Figure 10.4

La barre d'outils Débogage.



Exécuter pas à pas

Lorsqu'un programme ne produit pas le résultat escompté sans générer d'erreur, le premier test consiste à exécuter la procédure pas à pas, afin d'en examiner le déroulement et les conséquences sur le document, instruction après instruction.

Pour exécuter un programme VBA pas à pas :

1. Placez le curseur dans la procédure à exécuter, puis réduisez la fenêtre de Visual Basic Editor afin de visualiser à la fois la fenêtre de l'application hôte et la fenêtre Code de la procédure à tester.
2. Cliquez sur le bouton Pas à pas détaillé de la barre d'outils Débogage ; ou choisissez la commande Pas à pas détaillé du menu Débogage, ou appuyez sur la touche F8. La première instruction de la procédure est mise en évidence dans la fenêtre Code, tandis que le bouton Arrêt de la barre d'outils Standard apparaît estompé, indiquant que la procédure est interrompue en cours d'exécution.
3. Tapez de nouveau sur la touche F8. L'instruction précédemment mise en évidence dans la fenêtre Code s'exécute, tandis que l'instruction suivante est mise en évidence (voir Figure 10.5).

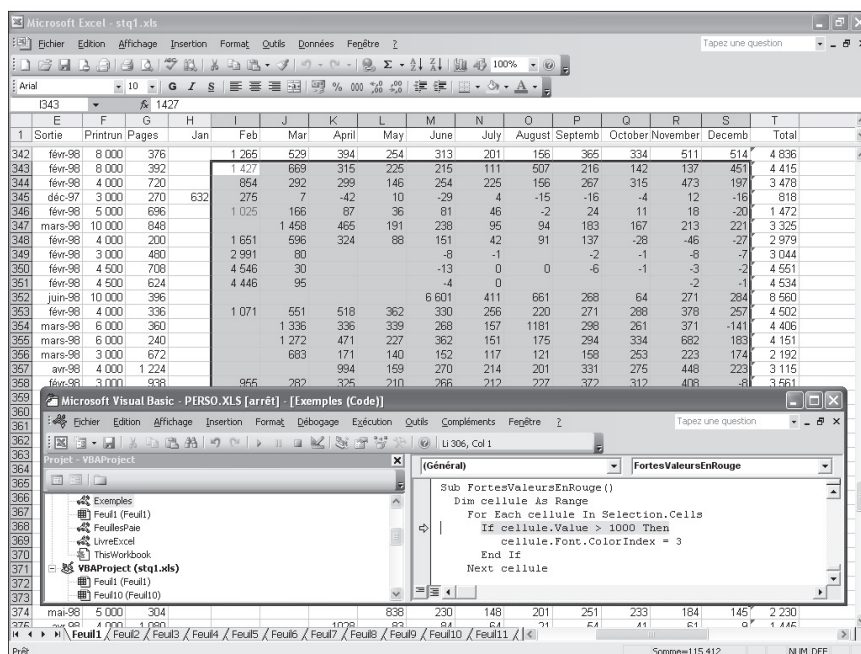


Figure 10.5

Exécutez les procédures pas à pas pour visualiser les conséquences de chacune des instructions.

4. Répétez l'opération de façon à visualiser l'incidence de chacune des instructions sur le document.



5. Vous pouvez aussi "lâcher" l'exécution de la procédure en cliquant sur le bouton Continuer de la barre d'outils Standard. L'exécution du programme se poursuit alors normalement à partir de l'instruction suivant la dernière instruction exécutée.



Vous pouvez aussi exécuter le code procédure par procédure, afin d'étudier la façon dont les procédures s'appellent dans le programme. Utilisez pour cela la commande Pas à pas principal. La commande Pas à pas sortant permet d'exécuter tout le code restant dans la procédure en cours. L'exécution est alors interrompue sur l'instruction suivant l'instruction d'appel de la procédure appelante.

La fenêtre Variables locales

Lorsque vous exécutez une procédure pas à pas, elle est en mode Arrêt. Vous pouvez alors visualiser la valeur des variables et des constantes aux différents stades de l'exécution du programme.



Le mode Arrêt désigne l'état d'une procédure dont l'exécution est interrompue. Cela est dû à une erreur d'exécution, à l'exécution pas à pas d'une procédure, à la rencontre d'une instruction End ou Stop dans la procédure, ou à l'interruption manuelle de l'exécution d'une procédure.

Pour visualiser la valeur d'une variable, activez l'option Info-bulles automatiques (Outils > Options). Lorsque le curseur est placé au-dessus d'une variable, la valeur de celle-ci apparaîtra dans une bulle d'aide (voir Figure 10.6).

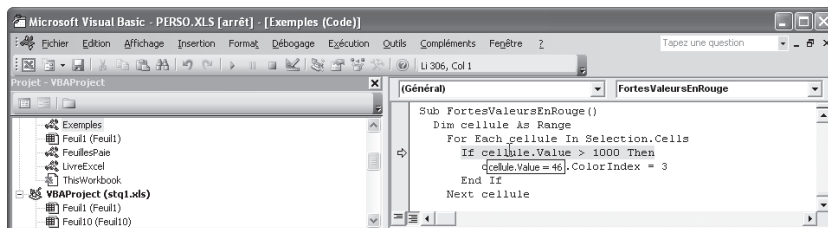


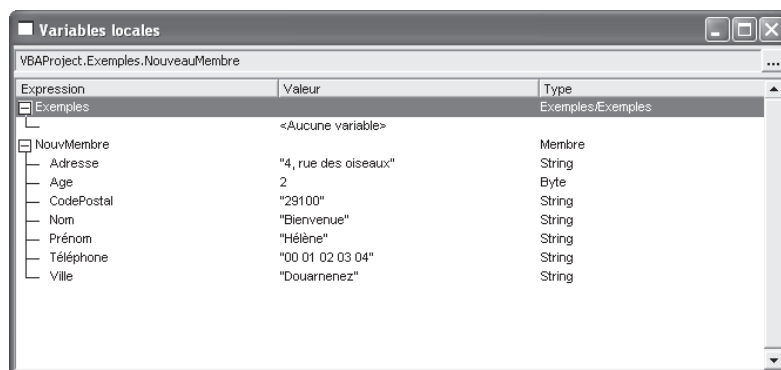
Figure 10.6

Les info-bulles automatiques permettent de connaître la valeur des variables à un moment précis de l'exécution d'un programme.



La fenêtre Variables locales permet d'obtenir des informations précises sur toutes les variables visibles à un moment donné de l'exécution du programme – nom, type et valeur des variables. Choisissez pour cela Affichage > Variables locales, ou cliquez sur le bouton Variables locales de la barre d'outils Débogage. La liste des variables visibles s'affiche (voir Figure 10.7).

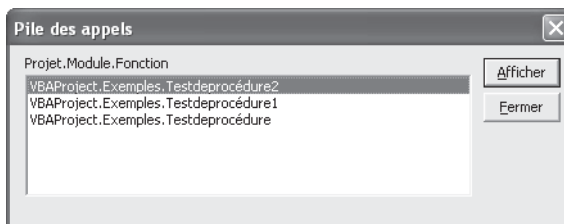
Figure 10.7
La fenêtre Variables locales fournit des informations complètes sur les variables.



Dans le haut de la fenêtre, le nom de la procédure en cours d'exécution s'affiche – ici, la procédure Testdeprocédure1, stockée dans le module Module1 du projet VBAProject. Les variables de niveau module apparaissent sous le nom du module – ici sous Module1 –, tandis que les variables de niveau procédure sont affichées telles quelles – ici TypePres et CodePostal.

Si la procédure en cours a été appelée, vous pouvez visualiser les variables des procédures appelantes. Cliquez sur le bouton Pile des appels. La fenêtre Pile des appels s'ouvre et affiche les appels de procédure actifs. Cliquez sur la procédure dont vous souhaitez visualiser les valeurs, puis sur Afficher. Vous pouvez voir à la Figure 10.8 que la procédure Testdeprocédure2 en cours a été appelée par la procédure Testdeprocédure1, elle-même appelée par la procédure Testdeprocédure. Lorsqu'une procédure appelée rend la main à la procédure appelante, elle disparaît de la pile des appels.

Figure 10.8
La pile des appels affiche les appels de procédure actifs.



Vous pouvez modifier la valeur des variables dans la fenêtre Variables locales, afin de tester le comportement du programme dans d'autres circonstances. Double-cliquez sur la valeur à modifier, puis saisissez la valeur voulue. Si la valeur choisie est incompatible avec le type de la variable, un message d'erreur s'affiche et la valeur de la variable reste inchangée.

Les points d'arrêt

Les points d'arrêt permettent d'interrompre l'exécution d'un programme sur une instruction précise. Cette possibilité est particulièrement intéressante lorsque vous soupçonnez l'origine d'une erreur. Vous pouvez ainsi exécuter normalement toutes les instructions ne posant pas de problème et définir un point d'arrêt pour une instruction dont vous n'êtes pas sûr. Une fois l'exécution interrompue, vous pouvez la poursuivre pas à pas, examiner la valeur des variables, etc.



Pour placer un point d'arrêt, placez le curseur sur l'instruction voulue et choisissez la commande Basculer le point d'arrêt du menu Débogage, ou cliquez sur le bouton Point d'arrêt de la barre d'outils, ou appuyez sur la touche F9. Vous pouvez aussi cliquer dans la marge de la fenêtre Code, en face de l'instruction voulue. Par défaut, l'instruction sur laquelle un point d'arrêt a été défini apparaît sur un arrière-plan de couleur bordeaux et un indicateur est placé en marge (voir Figure 10.9).

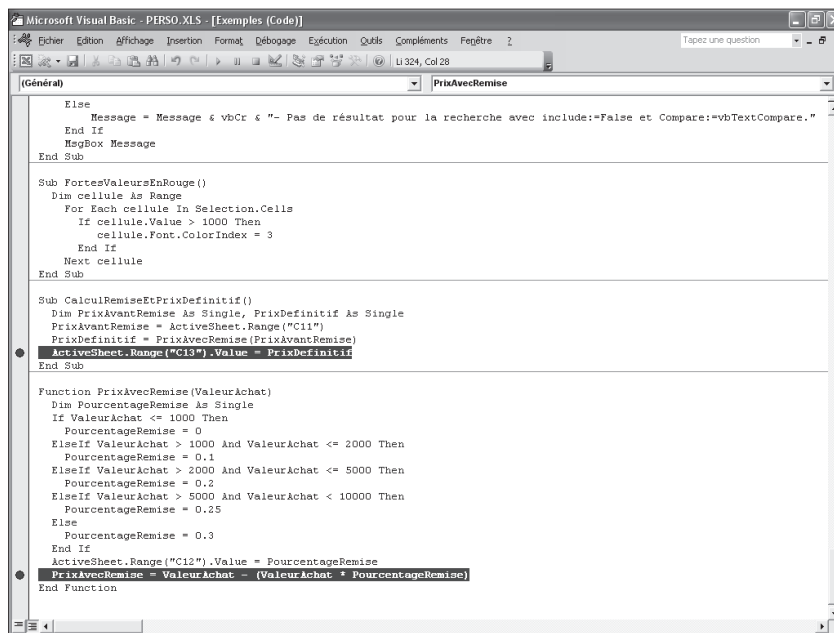


Figure 10.9

Les points d'arrêt permettent de définir des interruptions dans l'exécution du code.

Vous pouvez placer plusieurs points d'arrêt dans le code, afin de vérifier l'état des variables ou du document à différents stades de l'exécution, sans avoir à exécuter le programme pas à pas. Pour supprimer un point d'arrêt, procédez de la même façon que pour placer un point d'arrêt. Pour supprimer tous les points d'arrêt d'un module, sélectionnez la commande Effacer tous les points d'arrêt du menu Débogage ou tapez le raccourci clavier.



Lorsque vous quittez Visual Basic Editor, les points d'arrêt ne sont pas enregistrés. Pour conserver des points d'interruption du programme lors de la fermeture de Visual Basic Editor, utilisez l'instruction Stop, qui entraîne le passage de l'exécution d'une procédure en mode Arrêt.

Modifier l'ordre d'exécution des instructions

En mode Arrêt, vous pouvez à tout moment définir l'instruction suivante à exécuter dans une procédure. Il peut s'agir d'une instruction précédant l'instruction actuelle ou au contraire d'une instruction à venir. Pour modifier l'instruction suivante, placez le curseur dans l'instruction à exécuter et sélectionnez la commande Définir l'instruction suivante du menu Débogage, ou faites glisser l'indicateur de marge vers l'instruction voulue. Le code intermédiaire est alors ignoré et la procédure se poursuit à partir de l'instruction définie.

Vous pouvez ainsi ignorer une série d'instructions ne vous intéressant pas dans le cadre du débogage, ou au contraire éviter les instructions générant une erreur, pour étudier le comportement du programme dans d'autres circonstances. Vous pouvez alors utiliser la fenêtre Exécution pour exécuter des instructions n'apparaissant pas dans votre code, mais que vous envisagez de substituer aux instructions ignorées.



L'instruction suivante ne peut être définie qu'à l'intérieur de la procédure en cours.

La fenêtre Exécution

En mode Arrêt, la fenêtre Exécution permet d'exécuter tout type d'instruction qui ne se trouve pas dans le code du programme. Il suffit d'y écrire l'instruction à exécuter et d'appuyer sur Entrée. L'instruction s'exécute alors comme si elle faisait partie intégrante du code. Vous pouvez ainsi interroger ou modifier la valeur d'une variable ou d'une propriété, exécuter une instruction, etc.

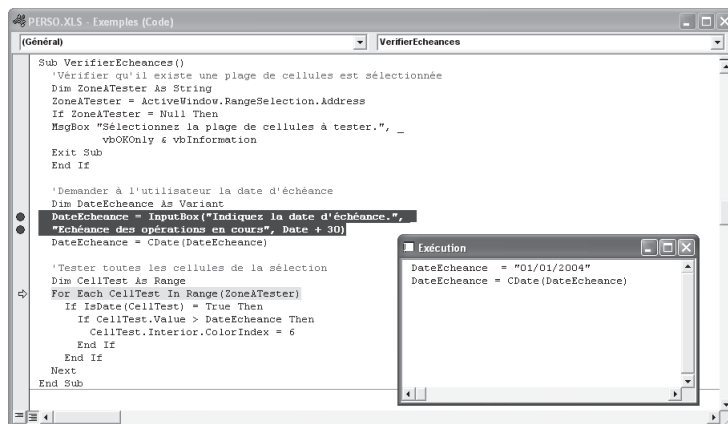


Pour afficher la fenêtre Exécution, choisissez la commande Fenêtre Exécution, ou cliquez sur le bouton Fenêtre Exécution de la barre d'outils Débogage, ou encore saisissez le raccourci clavier Ctrl+G. À la Figure 10.10, la valeur indiquée par l'utilisateur et affectée à la

variable DateEcheance a généré une erreur. Nous avons placé un point d'arrêt sur l'instruction ayant généré une erreur, afin d'y revenir par la suite. Nous avons utilisé la commande Définir l'instruction suivante du menu Débogage pour passer à l'instruction suivante et poursuivre l'exécution du code. Enfin, nous avons redéfini la valeur de DateEcheance à "01/04/98" et nous avons exécuté de nouveau l'instruction ayant généré l'erreur dans la fenêtre Exécution.

Figure 10.10

La fenêtre Exécution permet d'exécuter une instruction comme si elle faisait partie intégrante du code.



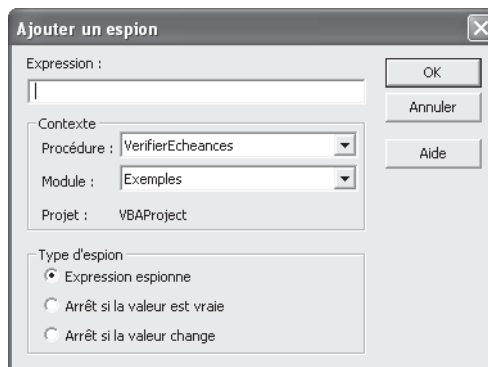
Les espions

Les espions permettent d'espionner les valeurs de variables ou de toute expression renvoyant une valeur dans un contexte déterminé. Pour créer un espion, procédez ainsi :

1. Choisissez la commande Ajouter un espion du menu Débogage. La boîte de dialogue Ajouter un espion s'affiche à l'écran (voir Figure 10.11).

Figure 10.11

La boîte de dialogue Ajouter un espion.



2. Dans la zone de texte Expression, saisissez l'expression dont vous souhaitez espionner la valeur.
3. Dans la zone Contexte, déterminez le contexte dans lequel l'expression sera espionnée. Par défaut les zones Procédure et Module affichent respectivement la procédure et le module en cours. Vous pouvez choisir d'espionner la valeur d'une expression dans une procédure d'un module, dans toutes les procédures d'un module, ou encore dans tous les modules. Il suffit, en général, de limiter la portée des espions à la portée des expressions espionnées.
4. Dans la zone Type d'espion, choisissez l'une des trois options disponibles :
 - **Expression espionne.** En mode Arrêt, la valeur en cours de l'expression s'affiche dans la fenêtre Espions.
 - **Arrêt si la valeur est vraie.** La procédure passe en mode Arrêt si la valeur de l'expression est définie à True.
 - **Arrêt si la valeur change.** L'exécution du code s'interrompt si la valeur de l'expression est définie à True.
5. Cliquez sur OK pour valider.

Vous pouvez aussi créer des espions en sélectionnant la variable ou l'expression voulue et en la faisant glisser dans la fenêtre Espions.



Pour afficher la fenêtre Espions, sélectionnez la commande Fenêtre Espions du menu Affichage ; ou cliquez sur le bouton Fenêtre Espions de la barre d'outils Débogage. Sept espions ont été placés dans la fenêtre Espions représentée à la Figure 10.12.

Figure 10.12
*Placez des espions
dans votre code...*

Espions			
Expression	Valeur	Type	Contexte
6d Application.UserName	<Hors du contexte>	Empty	Exemples.OuvertureSession
6d MotDePasse	<Hors du contexte>	Empty	Exemples.OuvertureSession
6d Utilisateur	<Hors du contexte>	Empty	Exemples.OuvertureSession
6d CellTest.Interior.ColorIndex	<Variable objet ou variable de bloc With non définie>	Variant/Integer	Exemples.VerifierEcheances
6d DateEcheance	04/12/2003	Variant/Date	Exemples.VerifierEcheances
6d Range(ZoneATester)		Object/Range	Exemples.VerifierEcheances
6d ZoneATester	"\$U\$356"	String	Exemples.VerifierEcheances

La fenêtre Espions contient quatre champs :

- **Expression.** Affiche l'expression espionnée. Il peut s'agir d'un nom de variable ou de toute expression renvoyant une valeur.
- **Valeur.** Affiche la valeur actuelle de l'expression espionnée en mode Arrêt. Dans le cas d'une expression de niveau procédure, cette zone affiche <Hors du contexte> si la procédure ne fait pas partie des appels de procédure actifs (elle n'apparaît alors pas dans la pile des appels).

Vous pouvez modifier la valeur d'une expression dans la fenêtre Espions, afin de tester le comportement du programme dans d'autres circonstances. Double-cliquez sur la valeur à modifier, puis saisissez la valeur voulue. Si la valeur choisie est incompatible avec l'expression, un message d'erreur s'affiche et la valeur de l'expression reste inchangée.

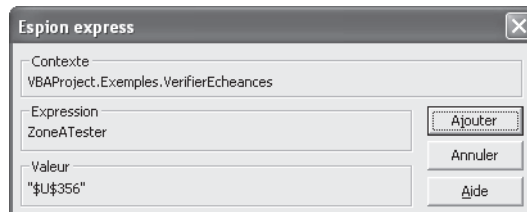
- **Type.** Affiche le type de la variable ou de la valeur renvoyée par l'expression. Si l'instruction en cours est hors du contexte d'espionnage, cette zone affiche Empty ou Variant/Empty.
- **Contexte.** Affiche le contexte défini pour l'expression espionne. À la Figure 10.12, vous pouvez constater que la première expression espionne est définie pour toutes les procédures du module Module1, tandis que les autres espions sont limités à une procédure spécifique du module.

Pour supprimer un espion, cliquez du bouton droit dans la fenêtre Espions et choisissez la commande Supprimer un espion du menu contextuel, ou sélectionnez-le et appuyez sur Suppr.



Si vous souhaitez interroger la valeur d'une expression pour laquelle vous n'avez pas placé d'espion, vous pouvez faire appel à l'Espion express. Sélectionnez l'expression voulue, puis choisissez la commande Expression express du menu Débogage. Vous pouvez aussi cliquer sur le bouton Espion express de la barre d'outils Débogage, ou encore utiliser Maj+F9. La boîte de dialogue Espion express vous renseigne alors sur le contexte de l'expression sélectionnée et sur sa valeur (voir Figure 10.13). Pour intégrer cette expression à la fenêtre Espions, cliquez sur Ajouter.

Figure 10.13
L'Espion express :
toujours prêt.



La pile des appels

La boîte de dialogue Pile des appels recense les appels de procédure actifs en mode Arrêt. Lorsqu'une procédure est exécutée, elle est ajoutée à la liste des appels de procédures actifs. Chaque fois qu'une procédure en appelle une autre, cette dernière est ajoutée en haut de la liste. Lorsqu'une procédure appelée s'achève, elle est supprimée de la pile des appels. La pile des appels recense donc toutes les procédures en cours d'exécution, selon leur ordre d'appel.

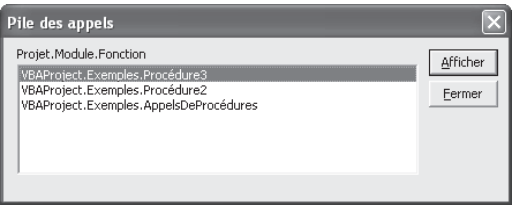


Il peut être intéressant de visualiser la pile des appels lors du débogage d'un programme VBA. Vous avez ainsi une idée précise des procédures en cours d'exécution et des appels successifs. Pour afficher la pile des appels, choisissez la commande Pile des appels du menu Affichage, ou cliquez sur le bouton Pile des appels de la barre d'outils Débogage, ou encore tapez le raccourci clavier Ctrl+L.

Vous pouvez voir à la Figure 10.14 que la procédure en cours d'exécution est Procédure3, qui a été appelée Procédure2, elle-même appelée par AppelsDeProcédures.

Figure 10.14

La boîte de dialogue Pile des appels.



Exemple de débogage

Nous allons créer ici un programme que nous déboguons jusqu'au moment où nous atteindrons une version fiable. Supposons que vous possédez un classeur Excel contenant de nombreuses données, mais dans lequel certaines lignes sont vides. Vous décidez donc d'écrire une macro VBA ayant pour fonction de supprimer les lignes vides. Nous supposons ici que lorsqu'une cellule de la colonne A ne contient pas de données, la ligne est vide et doit être supprimée. Ainsi, les lignes 7, 10, 11 et 16 du classeur représenté à la Figure 10.15 doivent être supprimées.

Figure 10.15

Le programme devra supprimer les lignes ne contenant pas de données.

	A	B	C	D	E
1	traffic figures from 26th february to 5th march 2002				
2		DE	FR	UK	ES
3	26-févr	22 480	52 634	5 922	98 498
4	27-févr	34 423	46 889	4 533	45 867
5	28-févr	32 279	40 978	4 188	27 678
6	01-mars	29 628	51 158	3 267	87 649
7					
8	02-mars	30 877	33 545	5 168	68 795
9	03-mars	26 647	31 143	5 056	57 869
10					
11					
12	04-mars	22 500	34 837	5 777	40 985
13	05-mars	25 800	36 885		36 885
14	Total	224 634	328 069	33 911	
15	average per day	28 079	41 009	4 844	
16					
17	average per month	842 378	1 230 259	145 333	
18	Traffic increase evaluation				
19	30% more with portal integration	1 120 362	1 636 244	193 293	

Une démarche logique consiste à commencer par définir la zone devant être traitée (de la cellule A1 à la dernière cellule de la colonne A ne contenant pas de données) par la macro et à affecter celle-ci à une variable objet. Nous utiliserons ensuite une structure For Each...Next pour vérifier la valeur de chacune des cellules de la zone préalablement définie, et supprimer la ligne correspondante chaque fois qu'une cellule ne contient pas de données. Le programme correspondant se présente ainsi :

```
1: Sub SupprLignesVidesBoguée()  
2:   'Définir la plage devant être testée  
3:   'On commence par définir la dernière cellule  
4:   'de la colonne A contenant des données  
5:   Dim MaPlage As Range  
6:   Set MaPlage = Range("A1")  
7:   While Range(MaPlage.End(xlDown).Address(rowabsolute:=False,  
       columnabsolute:=False)).Value <> ""  
8:       Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False,  
       columnabsolute:=False))  
9:   Wend  
10:  Set MaPlage = Range("A1:" & MaPlage.Address(rowabsolute:=False,  
       columnabsolute:=False))  
11:  
12:  'Supprimer les lignes ne contenant pas de données  
13:  Dim Cellule As Range  
14:  For Each Cellule In MaPlage  
15:      If Cellule.Value = "" Then  
16:          Rows(Cellule.Row).Delete  
17:      End If  
18:  Next Cellule  
19: End Sub
```

Lignes 5 à 10, la plage de cellules devant être testées est définie et affectée à la variable MaPlage de type Range. On commence par déclarer la variable MaPlage et par lui affecter la cellule A1 (lignes 5 et 6). Lignes 7 à 9, une structure While...Wend sert à définir la dernière cellule de la colonne A contenant des données. MaPlage se voit affecter la cellule renvoyée par la propriété End (ligne 8), tant que celle-ci n'est pas vide (ligne 7). Lorsque la propriété End renvoie une cellule vide, c'est que la cellule affectée à MaPlage est la dernière cellule contenant des données. La boucle While...Wend prend alors fin. Ligne 10, MaPlage reçoit la plage de cellules allant de la cellule A1 à la dernière cellule non vide de la colonne A, précédemment définie.

Lignes 14 à 18, une instruction For Each...Next est utilisée pour tester chacune des cellules de MaPlage. Une instruction conditionnelle sert à vérifier si la cellule est vide. Si tel est le cas, la ligne correspondante est supprimée (ligne 16).

Recherche du bogue

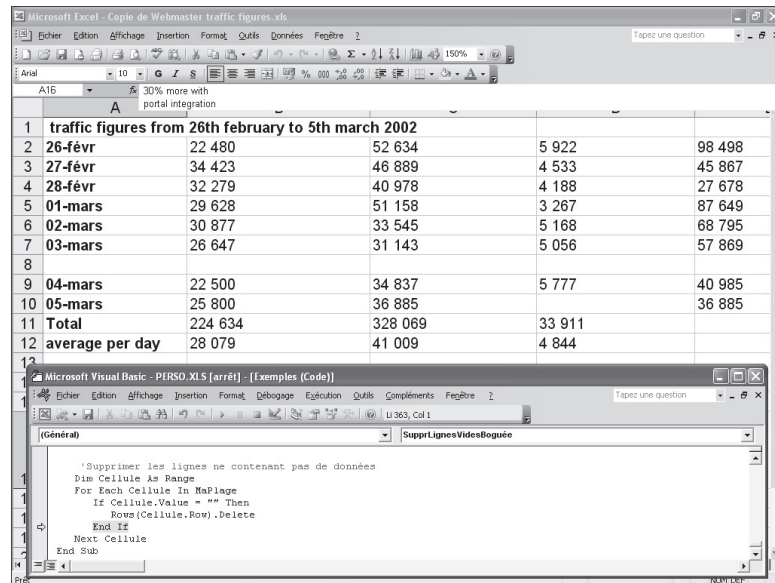
Exécutez le programme sur un classeur Excel contenant des lignes vides. Vous constatez que, s'il existe deux lignes vides consécutives, la deuxième n'est pas supprimée. Pour comprendre d'où vient le problème, procédez comme suit :

1. Placez un point d'arrêt avant la partie du programme qui semble générer l'erreur. Le problème ne vient visiblement pas de la première partie du programme, puisque la plage de cellules à tester est correctement définie. Placez donc un point d'arrêt devant l'instruction de la ligne 14, afin d'interrompre l'exécution du programme avant qu'il ne commence à supprimer les lignes.
2. Exécutez ensuite la macro. Lorsque l'instruction de la ligne 14 est atteinte, la macro s'interrompt et Visual Basic Editor passe au premier plan. L'instruction contenant le point d'arrêt est en surbrillance, et le bouton Arrêt de la barre d'outils Standard est estompé, indiquant que la macro est en mode Arrêt.
3. Réduisez la fenêtre de Visual Basic Editor de façon à voir le classeur Excel au second plan. Poursuivez ensuite l'instruction de la macro pas à pas (touche F8). Utilisez la barre de défilement d'Excel pour afficher les parties du classeur masquées par la fenêtre de Visual Basic Editor chaque fois que cela semble nécessaire. Faites ensuite repasser la fenêtre de Visual Basic Editor au premier plan à l'aide des touches Alt+Tab.
4. Observez attentivement l'effet de chaque instruction sur le classeur, ainsi que les valeurs que prennent les différentes variables et expressions du programme. Placez pour cela le curseur au-dessus des expressions `Cellule.Value` et `Cellule.Row` de façon à afficher les info-bulles automatiques. Soyez particulièrement attentif lorsqu'une cellule vide est testée.
5. Vous devriez vous rendre compte que, lorsque la ligne 10 est supprimée, la ligne 11 prend sa place et que la cellule suivante testée est la cellule A11.

La boucle `For Each...Next` traite les cellules de la plage `MaPlage` une à une. La cellule A1 est testée, puis la cellule A2, etc., jusqu'à atteindre la dernière cellule de la plage de cellules `MaPlage`. Lorsqu'une ligne est supprimée, la ligne suivante prend sa place ; elle est donc ignorée par la procédure. Dans le cas du classeur représenté à la Figure 10.15, les cellules A10 et A11 sont vides. Les lignes correspondantes doivent donc être supprimées. Lorsque la procédure atteint la cellule A10, celle-ci est supprimée. Cela a pour conséquence de décaler toutes les cellules vers le haut. La cellule A11 passe alors en A10, la cellule A12 en A11, etc. La boucle `For Each...Next` traite alors la cellule suivante, soit la cellule A11. Le contenu de la cellule précédemment en A11 ne sera donc pas traité, puisque celle-ci est passée en A10.

Figure 10.16

Exécutez la macro pas à pas et observez attentivement ses effets sur les macros, ainsi que les valeurs que prennent les variables.



Résolution du bogue

Plusieurs solutions permettent de régler ce problème.

1. Vous pouvez intégrer une instruction contrôlant de nouveau le contenu de la cellule testée lorsqu'une ligne est supprimée. Remplacez pour cela les instructions conditionnelles des lignes 15 à 17 par les instructions suivantes :

```

If Cellule.Value = "" Then
    Dim LigneSuppr As Long
    LigneSuppr = Cellule.Row
    Rows(Cellule.Row).Delete
    If Range("A" & LigneSuppr) = "" Then
        Rows(Range("A" & LigneSuppr).Row).Delete
    End If
End If
  
```

Avant de supprimer une ligne, le numéro de la ligne est stocké dans la variable LigneSuppr. La ligne est ensuite supprimée. Une instruction conditionnelle imbriquée supprime à nouveau cette ligne si la cellule de la colonne A correspondante est encore vide. Le problème est ainsi réglé si deux lignes consécutives sont vides. Mais le bogue persiste dès qu'il existe plus de deux lignes consécutives vides.

2. Vous pouvez également utiliser une structure For...Next à la place de la structure For Each...Next. L'utilisation d'une procédure For...Next permettra de définir un déroulement de la boucle commençant par la dernière cellule plutôt que par la première. Le programme se présente alors ainsi (les modifications apparaissent en gras) :

```
Sub SupprLignesVidesVersion2()
    Dim MaPlage As Range
    Set MaPlage = Range("A1")
    While Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
columnabsolute:=False)).Value <> ""
        Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
columnabsolute:=False))
    Wend

    Dim DerLigne As Long
    DerLigne = MaPlage.Row
    'Supprimer les lignes ne contenant pas de données
    Dim Compteur As Long
    For Compteur = DerLigne To 1 Step -1
        If Range( A & Compteur).Value =    Then
            Range( A & Compteur).Delete
        End If
    Next Compteur
End Sub
```

3. On pourra également stocker la liste des cellules vides dans une variable de matrice, et ne procéder à la suppression des lignes vides qu'une fois la liste des lignes à supprimer définie. On devra là aussi supprimer les cellules en partant de la dernière jusqu'à atteindre la première.

Option Base 1

```
1: Sub SupprLignesVidesVersion3()
2:     'Création d'une variable de matrice redimensionnable
3:     Dim MonTableau()
4:
5:     'Définir la plage devant être testée
6:     Dim MaPlage As Range
7:     Set MaPlage = Range("A1")
8:     While Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
columnabsolute:=False)).Value <> ""
9:         Set MaPlage = Range(MaPlage.End(xlDown).Address(rowabsolute:=False,
columnabsolute:=False))
10:    Wend
```

```
11: Set MaPlage = Range("A1:" & MaPlage.Address(rowabsolute:=False,
    columnabsolute:=False))
12:
13: 'On stocke dans la variable de matrice les valeurs
14: 'de toutes les lignes devant être supprimées
15: Dim Cellule As Range
16: For Each Cellule In MaPlage
17:     If Cellule.Value = "" Then
18:         'Une erreur sera générée à l'appel de la fonction
19:         'UBound si le tableau n'a encore reçu aucune valeur
20:         On Error Resume Next
21:         ReDim Preserve MonTableau(UBound(MonTableau) + 1)
22:         'L'instruction conditionnelle suivante gère cette erreur
23:         If Err.Number = 9 Then
24:             ReDim MonTableau(1)
25:             Err.Clear
26:         End If
27:         MonTableau(UBound(MonTableau)) = Cellule.Row
28:     End If
29: Next Cellule
30:
31: 'Suppression des lignes vides
32: Dim Compteur As Single
33: For Compteur = UBound(MonTableau) To LBound(MonTableau) Step -1
34:     Rows(MonTableau(Compteur)).Delete
35: Next Compteur
36: End Sub
```

L'instruction `Option Base 1` a été placée dans la zone de déclarations du module, de façon que la première valeur d'index des tableaux soit 1 et non 0.

Ligne 3, une variable tableau redimensionnable est déclarée. Elle servira à stocker les numéros des lignes à supprimer. Lignes 6 à 11, la zone à traiter est définie et affectée à la `MaPlage`, de la même façon que dans les versions précédentes du programme.

Lignes 15 à 29, les numéros des lignes à supprimer sont définis et stockés dans `MonTableau`. La variable `Cellule` de type `Range` est tout d'abord déclarée. Une structure `For Each... Next` sert ensuite à tester l'ensemble des cellules de `MaPlage` (lignes 16 à 29). À chaque passage de la boucle, une structure conditionnelle (lignes 17 à 28) évalue la valeur de la cellule testée et stocke le numéro de la ligne à la variable de matrice `MonTableau` si la cellule est vide. La variable est tout d'abord redimensionnée (ligne 21). On se sert pour cela de la fonction `UBound` qui renvoie la taille du tableau, à laquelle on ajoute 1. Ligne 27, le dernier espace de stockage de `MonTableau` – `MonTableau(UBound(MonTableau))` – reçoit le numéro de la ligne de la cellule testée.

Rappel

Notez l'utilisation du mot clé Preserve lors du redimensionnement du tableau (ligne 21). Celui-ci est indispensable pour que le tableau ne soit pas réinitialisé.

Lignes 20 à 26, un gestionnaire d'erreur a été mis en place. En effet, la première fois que la fonction UBound est utilisée (ligne 21), une erreur est générée. L'instruction Resume Next de la ligne 20 force le passage à l'instruction suivante en cas d'erreur. Ligne 23, on interroge la valeur de la propriété Number l'objet Err pour vérifier si une erreur a été générée. Si tel est le cas (si le tableau n'a pas encore été dimensionné), MonTableau est dimensionné avec un espace de stockage et l'objet Err est réinitialisé à l'aide de la méthode Clear.

Conseil

Pour connaître le numéro d'une erreur (la valeur de la propriété Number de l'objet Err), générez volontairement cette erreur et relevez le numéro d'erreur indiquée dans la boîte de dialogue Visual Basic affichée au moment où l'erreur survient.

Lignes 32 à 35, les lignes vides sont supprimées. On utilise pour cela une boucle For... Next dont le compteur commence à la valeur d'index la plus importante de MonTableau pour atteindre la valeur la plus basse, et est décrémenté de 1 à chaque passage. À chaque passage de la boucle, l'instruction de la ligne 34 supprime la ligne dont le numéro correspond à la valeur stockée dans l'espace de stockage d'index Compteur de MonTableau.

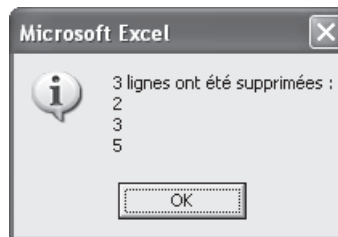
Info

L'intérêt de cette version du programme est de permettre de stocker les lignes supprimées dans une variable. Placez les instructions suivantes en fin de programme pour afficher le nombre de lignes supprimées ainsi que la liste de ces dernières :

```
Dim Message As String
Message = UBound(MonTableau) & " lignes ont été supprimées : "
For Compteur = 1 to UBound(MonTableau)
    Message = Message & vbCrLf & MonTableau(Compteur)
Next Compteur
MsgBox Message, vbOKOnly + vbInformation
```

Figure 10.17

Cette version du programme permet de connaître les lignes supprimées.



Une telle méthode se révélera particulièrement intéressante si vous créez une procédure destinée à supprimer des lignes contenant des informations, mais que vous souhaitez récupérer ces informations dans un autre classeur Excel. On utilisera alors une variable de matrice dynamique multidimensionnelle. Celle-ci aura deux dimensions : l'une correspondant aux lignes du classeur, l'autre correspondant aux colonnes. La taille de la première dimension (le nombre de lignes à supprimer ou le nombre de colonnes du classeur contenant des données à conserver) devra être définie avant de stocker les données dans la variable – seule la dernière dimension pouvant être redéfinie en conservant les valeurs de la variable.

Gestion des erreurs et des exceptions

Un programme VBA peut s'exécuter correctement dans la plupart des cas et générer des erreurs d'exécution dans des contextes spécifiques. Une erreur peut, par exemple, être générée si l'utilisateur n'entre pas le type d'information attendue dans une boîte de dialogue. Une erreur sera également générée si le format d'une cellule ne correspond pas au type de données qu'un programme tente d'exploiter, ou encore si un programme tente de modifier un classeur Excel en cours d'utilisation. Nombre d'erreurs de ce type, liées à un code écrit pour un contexte particulier, sont susceptibles d'affecter une macro. Il est important de prévoir ce type d'erreur et de mettre en place un *gestionnaire d'erreur*, afin que la procédure contourne l'erreur et s'exécute normalement.



Un gestionnaire d'erreur est un ensemble d'instructions qui est censé permettre la poursuite de l'exécution d'une procédure. En général, il est appelé par un détecteur d'erreur, dont la fonction est de repérer l'erreur lorsqu'elle survient.

Pour détecter une éventuelle erreur, placez une instruction `On Error` devant l'instruction susceptible d'en générer une :

- **On Error Resume Next.** Ignore l'instruction ayant généré une erreur et passe à l'instruction suivante.
- **On Error GoTo étiquette.** Permet d'appeler un gestionnaire d'erreur repéré par l'étiquette.

Une technique courante consiste à placer le gestionnaire d'erreur à la fin de la procédure et à glisser une instruction `Exit` avant l'étiquette le délimitant :

```
Sub MaProcédure()  
...  
On Error GoTo GestionnaireErreur  
Instruction susceptibles de générer une erreur
```

```

...
Exit Sub
GestionnaireErreur:
    Instructions de gestion des erreurs
Resume
End Sub

```

Ainsi, dès qu'une erreur est détectée, le gestionnaire d'erreur sera appelé. L'instruction `Resume` ne peut être placée que dans un gestionnaire d'erreur. Elle rend la main à l'instruction ayant généré une erreur, qui s'exécute de nouveau. Si aucune erreur n'est générée, le programme se déroule normalement jusqu'à l'instruction `Exit Sub`. La procédure s'achève alors, sans que le gestionnaire d'erreur ait été exécuté.

Lorsqu'une erreur est générée, la propriété `Number` de l'objet `Err` se voit affecter une valeur identifiant le type d'erreur survenu. Après gestion de l'erreur, il est important de redéfinir cette propriété à 0 (pas d'erreur), afin que les éventuelles autres erreurs puissent être détectées. Utilisez alors l'instruction d'affectation `Err.Number = 0`.

Exemple de gestion d'erreur

La procédure suivante – étudiée au Chapitre 7 – générera une erreur si l'utilisateur n'entre pas une date dans un format valide dans la boîte de dialogue affichée par la fonction `InputBox` (voir Figure 10.15).

```

Sub VerifierEcheances()
    'Vérifier qu'il existe une plage de cellules est sélectionnée
    Dim ZoneATester As String
    ZoneATester = ActiveWindow.RangeSelection.Address
    If ZoneATester = Null Then
        MsgBox "Sélectionnez la plage de cellules à tester.", _
            vbOKOnly & vbInformation
    Exit Sub
End If

    'Demander à l'utilisateur la date d'échéance
    Dim DateEcheance As Variant
    DateEcheance = InputBox("Indiquez la date d'échéance.", _
        "Echéance des opérations en cours", Date + 30)
    If DateEcheance = "" Then
        Exit Sub
    End if
    DateEcheance = CDate(DateEcheance)

```



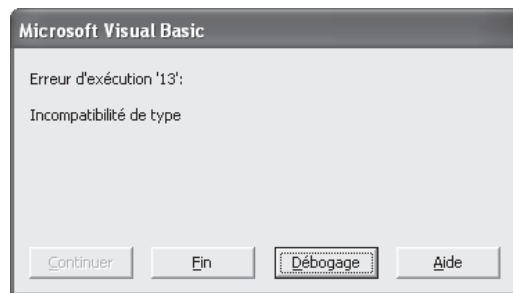
```

'Tester toutes les cellules de la sélection
Dim CellTest As Range
For Each CellTest In Range(ZoneATester)
    If IsDate(CellTest) = True Then
        If CellTest.Value > DateEcheance Then
            CellTest.Interior.ColorIndex = 6
        End If
    End If
End For
Next
End Sub

```

Figure 10.18

Cette erreur peut être gérée par le programme.



La procédure suivante gère cette erreur en faisant appel à un gestionnaire d'erreur qui affiche une nouvelle boîte de dialogue dans laquelle l'utilisateur est invité à entrer une date dans un format valide. L'instruction ayant généré l'erreur est alors répétée et le programme se déroule normalement, jusqu'à l'instruction `Exit Sub` qui entraîne la sortie de la procédure. Si l'utilisateur clique sur le bouton Annuler, la variable `DateEcheance` renverra `Empty` et l'instruction `Exit Sub` entraînera la sortie du programme.

```

Sub VerifierEcheances()
    Dim ZoneATester As String
    ZoneATester = ActiveWindow.RangeSelection.Address
    If ZoneATester = Null Then
        MsgBox "Sélectionnez la plage de cellules à tester.", _
            vbOKOnly & vbInformation
    Exit Sub
    End If

    Dim DateEcheance As Variant
    DateEcheance = InputBox("Indiquez la date d'échéance.", _
        "Echéance des opérations en cours", Date + 30)
    If DateEcheance = "" Then
        Exit Sub
    End If
End Sub

```

```
End if
On Error GoTo GestionnaireErreur
DateEcheance = CDate(DateEcheance)

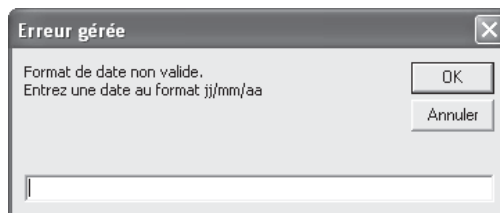
Dim CellTest As Range
For Each CellTest In Range(ZoneATester)
    If IsDate(CellTest) = True Then
        If CellTest.Value > DateEcheance Then
            CellTest.Interior.ColorIndex = 6
        End If
    End If
End For
Next

Exit Sub

GestionnaireErreur:
Err.Number = 0
DateEcheance = InputBox("Format de date non valide." & _
    Chr(10) & "Entrez une date au format jj/mm/aa", "Erreur gérée", _
    "jj/mm/aa")
If DateEcheance = Empty Then Exit Sub
Resume
End Sub
```

Figure 10.19

L'erreur est détectée et l'utilisateur est invité à entrer une information valide ou à annuler l'opération.





Intégrer des applications VBA dans l'interface d'Excel

Au sommaire de ce chapitre

- Affecter une macro à un raccourci clavier
- Affecter une macro à un bouton
- Affecter une macro à une commande de menu
- Supprimer une commande de menu
- Modifier l'image et/ou le nom d'une commande
- Affecter une macro à un objet

Vous pouvez très simplement améliorer l'accessibilité d'une macro en lui affectant un raccourci clavier, une ligne de commande dans un menu ou encore un bouton de barre d'outils.

Si vous n'utilisez que rarement une macro, contentez-vous de l'exécuter par la boîte de dialogue Macros. Cependant, la nature des macros ou la fréquence de leur utilisation peuvent justifier un accès plus rapide. Une macrocommande regroupant quelques commandes ne présente qu'un intérêt limité si elle ne peut être exécutée rapidement, par un raccourci clavier ou une icône (ou les deux). Il sera par contre préférable d'affecter une ligne de commande à un programme aux conséquences plus larges et d'une utilisation moins fréquente.

Affecter une macro à un raccourci clavier

Si vous n'avez pas affecté de raccourci clavier à une macro au moment de sa création, vous pouvez très simplement le faire après coup :

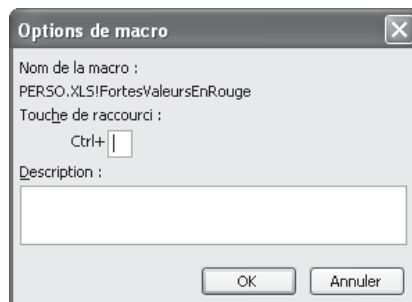
1. Cliquez sur le bouton Macros de l'onglet Développeur.
2. Dans la boîte de dialogue Macro, sélectionnez la macro voulue puis cliquez sur le bouton Options.

La boîte de dialogue Options de macro s'affiche à l'écran (voir Figure 11.1).

3. Indiquez le raccourci de votre choix en saisissant une lettre dans la zone conçue à cet effet. Vous pouvez aussi ajouter une description à la macro, si vous avez omis de le faire lors de la création.
4. Cliquez sur OK pour valider les modifications.

Figure 11.1

Vous pouvez aisément affecter un raccourci clavier à une macro.



Personnaliser le ruban et la barre d'outils Accès rapide

Avec Excel 2010, vous pouvez personnaliser tous les onglets du ruban ainsi que la barre d'outils Accès rapide (Excel 2007 ne permettait pas la personnalisation des onglets du ruban, seule la barre d'outils Accès rapide pouvait être modifiée). Pour ce faire, procédez comme suit :

1. Affichez le fichier PERSONAL.XLSB et cliquez sur l'onglet Fichier, puis sur Options. Dans la boîte de dialogue Options, choisissez Personnaliser le ruban ou Barre d'outils Accès rapide.
2. Dans la zone de liste Choisir les commandes dans les catégories suivantes, choisissez Macros (Figure 11.2). L'ensemble des macros accessibles s'affiche.

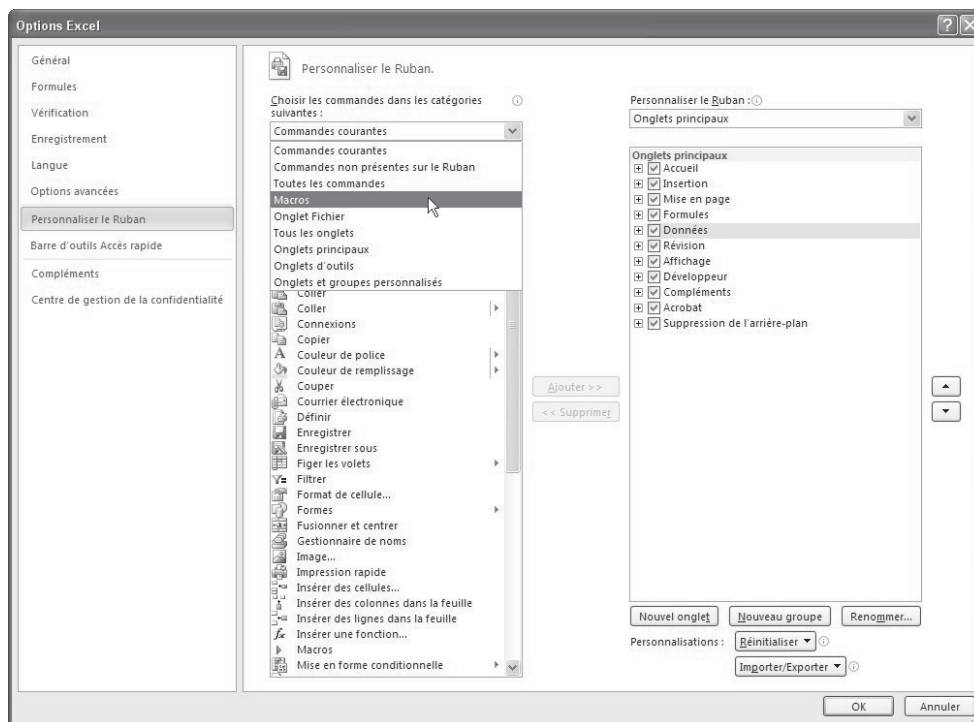


Figure 11.2

Avec Excel 2010, vous pouvez personnaliser la barre d'outils Accès rapide, mais aussi le ruban.

3. Dans la zone Personnaliser le ruban, sélectionnez l'une des trois options disponibles : Tous les onglets, Onglets principaux ou Onglets d'outils.
4. Choisissez l'élément que vous voulez personnaliser :
 - sélectionnez un des onglets de la liste, puis choisissez Nouveau groupe pour personnaliser un onglet existant ;
 - cliquez sur le bouton Nouvel onglet pour créer un nouvel onglet (comme dans notre exemple).
5. Cliquez droit sur le nouvel onglet créé, et choisissez Renommer. Dans la boîte de dialogue qui s'affiche, saisissez un nom approprié, ici Macros (voir Figure 11.3), puis validez. Procédez de même pour le libellé Nouveau groupe (Personnalisé), situé sous l'onglet que vous venez de créer.

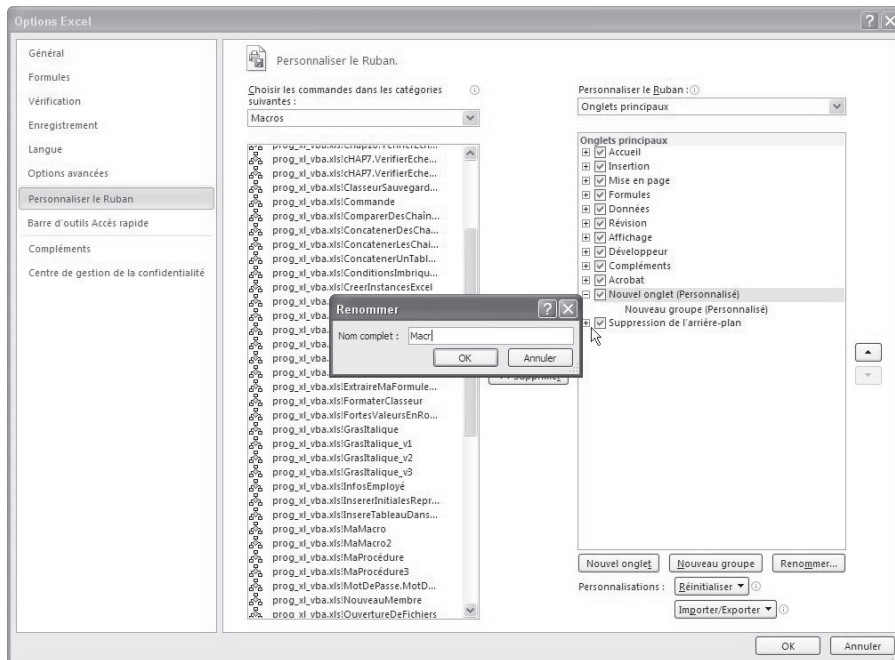


Figure 11.3

Si vous choisissez d'affecter vos macros à un nouvel onglet, donnez-lui un nom représentatif.

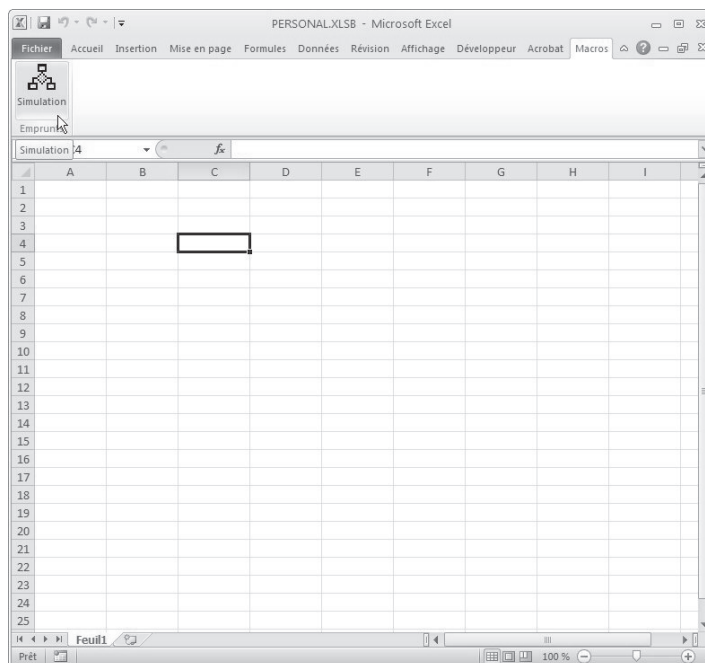
Astuce

Pour modifier l'ordre des onglets sur le ruban, sélectionnez l'onglet voulu et utilisez les flèches haut et bas pour repositionner l'onglet sélectionné.

6. Sélectionnez la macro de votre choix dans le volet gauche, puis cliquez sur Ajouter. Le nom de la macro apparaît dans le groupe sélectionné du nouvel onglet. Renommez la macro comme vous l'avez fait pour l'onglet et le groupe à l'étape 5. Validez.
7. Enfin, utilisez les flèches situées sur la droite de la fenêtre pour définir la position de votre commande sur la barre d'outils, puis validez.

Figure 11.4

La macro est maintenant accessible via le nouvel onglet.



Pour supprimer une commande personnalisée, cliquez dessus du bouton droit et choisissez la commande Supprimer dans le menu contextuel qui s'affiche.

Personnaliser les barres d'outils dans les versions antérieures à Excel 2007

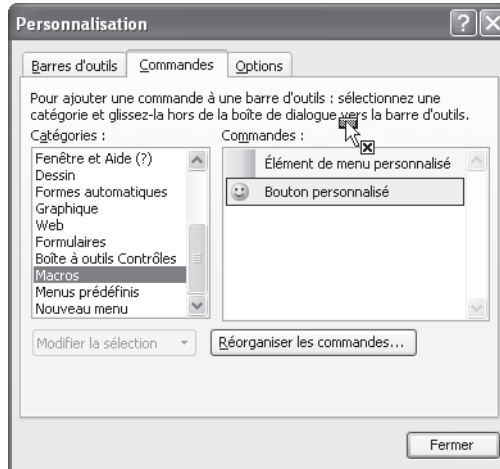
Cette section présente la méthode de personnalisation de l'interface d'Excel avec une version antérieure à 2007 :

1. Choisissez la commande Personnaliser du menu Outils et activez l'onglet Commandes de la boîte de dialogue Personnaliser.

2. Dans la liste Catégories, choisissez Macros. La liste Commandes de la boîte de dialogue Personnaliser affiche deux options : Élément de menu personnalisé et Bouton personnalisé.
3. Sélectionnez Bouton personnalisé et maintenez le bouton de la souris enfoncé. Un rectangle apparaît sur le pointeur, tandis qu'un × indique que l'emplacement du curseur ne permet pas de créer une commande (voir Figure 11.5).

Figure 11.5

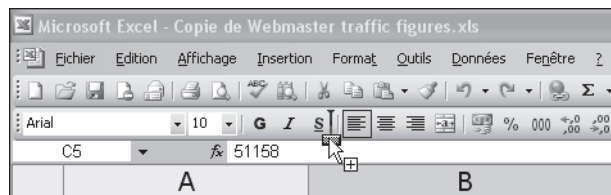
Faites glisser la macro à laquelle vous souhaitez affecter un bouton de barre d'outils.



4. Tout en maintenant le bouton de la souris enfoncé, placez le pointeur sur la barre d'outils voulue. Une barre verticale indique où sera placée la commande, tandis que le × se transforme en un + (voir Figure 11.6).

Figure 11.6

Une barre verticale indique où sera placée la commande.



5. Relâchez le bouton de la souris. Une icône représentant un Smiley s'affiche dans la barre d'outils.
6. Cliquez droit sur le nouveau bouton et choisissez la commande Affecter une macro. Dans la boîte de dialogue Affecter une macro qui s'affiche, sélectionnez la macro voulue et cliquez sur le bouton OK (voir Figure 11.7).

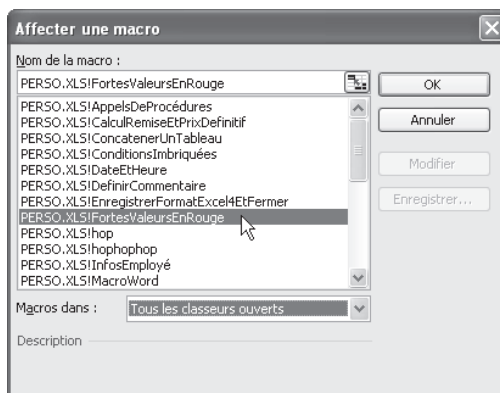
7. Reportez-vous à la section Modifier l'image et/ou le nom d'une commande pour déterminer l'icône à affecter au raccourci ainsi créé, puis cliquez sur le bouton Fermer de la boîte de dialogue Personnaliser.



Lorsque vous ajoutez un bouton à une barre d'outils Excel, ouvrez le menu contextuel du bouton personnalisé et affectez-lui un nom représentatif. Lorsque le pointeur est placé au-dessus du bouton, son nom s'affiche dans une info-bulle.

Figure 11.7

Sélectionnez la macro à affecter au nouveau bouton.



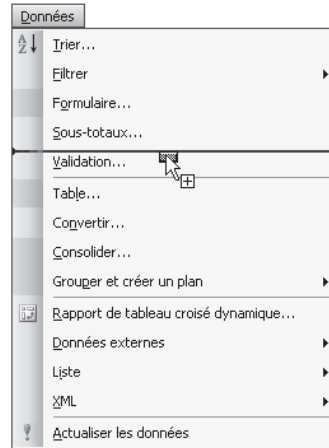
Personnaliser les menus dans les versions antérieures à Excel 2007

La procédure permettant d'affecter une commande de menu à une macro avec une version d'Excel antérieure à 2007 s'apparente à l'ajout d'un bouton sur une barre d'outils. Procédez ainsi :

1. Répétez les étapes 1 à 3 de la section précédente, en choisissant Élément de menu personnalisé à l'étape 3.
2. Tout en maintenant le bouton de la souris enfoncé, placez le pointeur sur le nom du menu dans lequel vous souhaitez placer la commande. Le menu se déroule.
3. Placez le curseur à l'endroit où vous souhaitez déposer la commande. Une barre horizontale vous indique précisément l'emplacement qu'elle prendra (voir Figure 11.8).

Figure 11.8

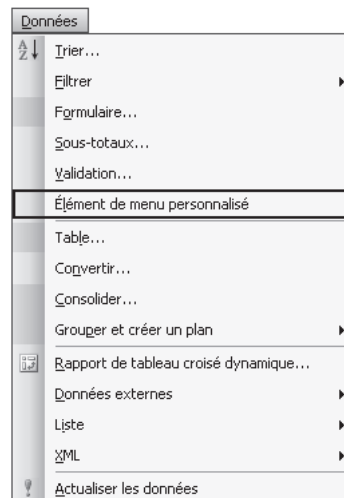
Une barre horizontale symbolise l'emplacement que prendra la commande.



4. Relâchez le bouton de la souris. Le libellé Élément de menu personnalisé apparaît dans le menu (voir Figure 11.9).
5. Reportez-vous à la section Modifier l'image et/ou le nom d'une commande pour modifier le nom de la commande et déterminer éventuellement l'icône à affecter au raccourci ainsi créé. Cliquez ensuite sur le bouton Fermer de la boîte de dialogue Personnaliser.

Figure 11.9

Il ne vous reste qu'à affecter un nom représentatif à la nouvelle commande.



Astuce

Il peut être judicieux de regrouper vos macros dans des menus qui leur soient propres. Pour créer un nouveau menu, sélectionnez Nouveau menu dans la zone Catégories de la boîte de dialogue Personnaliser, puis faites glisser la commande Nouveau menu à l'emplacement voulu sur la barre de menus.

Affecter une macro à un objet

Une macro peut également être affectée à un objet tel qu'un graphique Excel ou un dessin de la couche Dessin des applications Office. Pour cela, cliquez du bouton droit sur l'objet voulu et choisissez la commande Affecter une macro du menu contextuel.

Lorsqu'une macro est affectée à un objet, le curseur se transforme en main lorsqu'il est placé au-dessus de l'objet en question. Un clic sur celui-ci déclenche la macro.

Info

Pour manipuler les commandes de menu et les barres d'outils Excel par programmation – par exemple, ajouter ou supprimer des commandes selon le contexte –, utilisez la propriété CommandBars. Pour plus de précisions, consultez l'aide en ligne.



Développer des interfaces utilisateur

CHAPITRE 12. Créer des interfaces utilisateur

CHAPITRE 13. Exploiter les propriétés des contrôles ActiveX

CHAPITRE 14. Maîtriser le comportement des contrôles



12

Créer des interfaces utilisateur

Au sommaire de ce chapitre

- Les phases de développement de feuilles
- Créer une feuille
- Les contrôles de la boîte à outils
- Placer des contrôles sur une feuille
- Mise en forme des contrôles
- Personnaliser la boîte à outils
- Afficher/masquer une feuille

Les feuilles constituent un élément essentiel de la programmation Visual Basic. Les feuilles sont des zones sur lesquelles vous placez des contrôles ActiveX tels que des cases à cocher, des boutons d'option, des zones de texte, des boutons de commande, etc. Ils forment une interface graphique permettant une interaction simple et intuitive entre l'utilisateur final et le programme.

Les événements utilisateur, tels qu'un clic de souris ou une modification de la valeur d'un contrôle, qui touchent les contrôles sont automatiquement repérés par le programme. Il vous suffit d'associer du code à un événement donné pour que celui-ci s'exécute lorsque l'événement est repéré. On parle alors de *procédure événementielle* ou *procédure d'événement*.

Les contrôles prennent une valeur, déterminée par leur état (case cochée ou non cochée, texte d'une zone de texte, etc.). L'exploitation des feuilles consiste généralement à passer les informations entrées par l'utilisateur sur la feuille (les valeurs des différents contrôles) à une procédure de code lorsque l'utilisateur valide les informations fournies – en cliquant sur un bouton libellé OK, par exemple. Ces données sont ensuite exploitées par le programme.

Figure 12.1

Les feuilles constituent l'interface graphique de vos projets VBA.

Contrat de traduction

Conditions de traduction

Volume de traduction : 100 pages

Rémunération : 60 francs / page

Rémunération Audio : 10 francs / page

☒ Nombre de pages définitif

☐ Personnaliser le nombre de paiements

Options d'impression

☒ Imprimer les contrats

☒ Edition et audio en 2 exemplaires.

☐ Fiches de paie en 1 exemplaires.

Annuler Retour Suite

Les phases de développement de feuilles

Les feuilles permettent de ramener des tâches complexes à la simple information des champs d'une boîte de dialogue pour l'utilisateur final.

La création de feuilles VBA se réalise en trois phases :

- **Détermination des besoins.** Avant de vous lancer dans la création d'une feuille, réfléchissez aux fonctions que devra jouer l'interface développée. Quels doivent en être les différents contrôles ? Comment seront-ils organisés sur la feuille ? Quel type d'interaction entre les contrôles ? À quels événements utilisateur devront-ils répondre ? Nous vous conseillons de réaliser sur papier un dessin approximatif de la feuille et de noter les informations essentielles sur les différents contrôles la composant avant d'entamer la deuxième phase.
- **Création visuelle de la feuille.** Durant cette phase, vous allez dessiner votre feuille dans Visual Basic Editor. Vous créerez la feuille et y placerez les contrôles voulus (cases à cocher, boutons d'option). Vous paramètrerez les propriétés de la feuille et des contrôles qui la composent, afin d'en déterminer l'apparence (couleur, taille, texte par défaut) et le comportement (accessibilité, nombre de caractères autorisés, etc.).
- **Écriture du code attaché à la feuille.** Sans code affecté aux différents éléments qui la composent, une feuille n'est qu'une jolie boîte de dialogue sans fonctionnalité. Durant cette phase, vous déterminerez le comportement de la feuille face aux différents événements utilisateur pouvant l'affecter et écrirez le code permettant d'exploiter les données fournies par l'utilisateur.

Ce chapitre présente la création de feuilles. Vous y découvrirez les différents contrôles à votre disposition, apprendrez à les placer sur une feuille et à utiliser les outils de VBA pour créer des interfaces à l'apparence professionnelle. Les outils de développement visuel de feuilles de Visual Basic permettent de créer simplement et rapidement des interfaces complètes, semblables aux boîtes de dialogue des applications Office.

Créer une feuille

Le développement de feuilles se fait dans la fenêtre UserForm de Visual Basic Editor. Pour accéder à la fenêtre UserForm, vous devez créer une nouvelle feuille ou ouvrir une feuille existante.

Pour créer une nouvelle feuille :

1. Ouvrez l'Explorateur de projet et sélectionnez le projet dans lequel vous souhaitez créer une feuille.



2. Cliquez sur la flèche du bouton Ajouter de la barre d'outils Standard de Visual Basic Editor et sélectionnez UserForm ou sélectionnez la commande UserForm du menu Insertion ou cliquez du bouton droit sur n'importe quel élément du projet dans l'Explorateur de projet et, dans le menu contextuel qui s'affiche, sélectionnez Insertion > UserForm.
3. Une fenêtre UserForm s'ouvre (voir Figure 12.2). La feuille et la boîte à outils y sont affichées. Par défaut, la feuille est intitulée UserForm1, UserForm2 s'il existe déjà une feuille UserForm1, etc.

La nouvelle feuille apparaît dans le dossier Feuilles de l'Explorateur de projet.

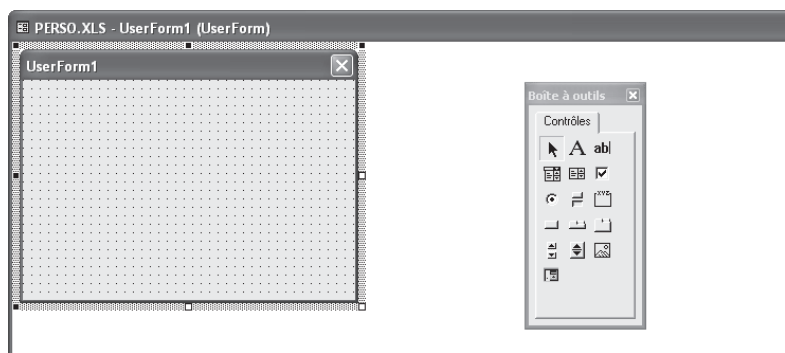


Figure 12.2

Votre feuille encore vierge de tout contrôle.

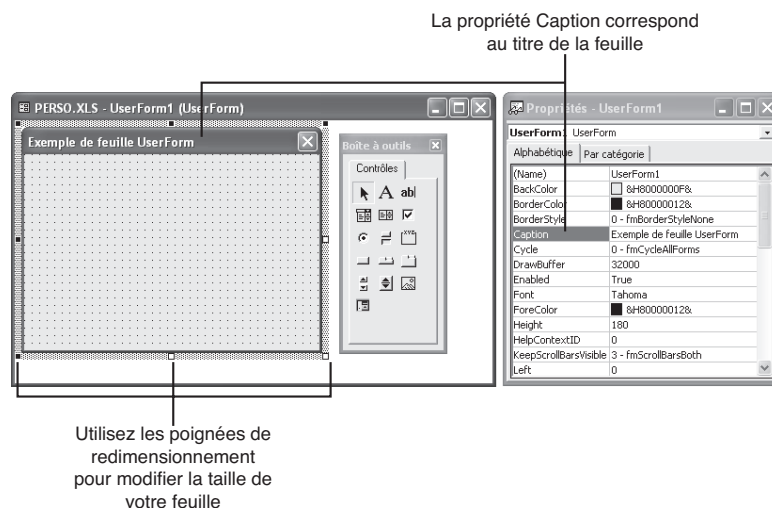
4. Ouvrez la fenêtre Propriétés de la feuille (F4) et attribuez-lui une propriété Name et une propriété Caption.

La propriété Name est le nom qui sera utilisé dans le code pour faire référence à cette feuille. La propriété Caption correspond au libellé de la feuille qui apparaît dans sa barre de titre. La Figure 12.3 présente une feuille dont la propriété Caption a été définie à "Exemple de feuille UserForm".

5. Vous pouvez à tout moment utiliser les poignées de redimensionnement pour modifier la taille de la feuille.
6. Cliquez sur le bouton Enregistrer de la barre d'outils Standard pour enregistrer votre projet.

Pour ouvrir la fenêtre UserForm d'une feuille existante, ouvrez l'Explorateur de projet (Ctrl+R) et double-cliquez sur celle-ci.

Figure 12.3
Définissez les propriétés
Name et Caption de la
feuille.



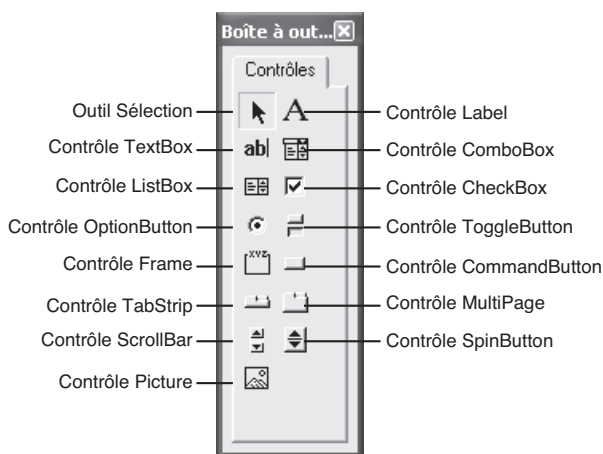
Les contrôles de la boîte à outils

La boîte à outils contient les contrôles que vous pouvez placer sur votre feuille. Au même titre que la feuille elle-même, les contrôles sont des objets. Vous n'avez pas à vous soucier de la façon dont les contrôles fonctionnent. Il vous suffit d'en connaître les méthodes, propriétés et événements membres pour pouvoir les exploiter.



Pour afficher ou masquer la boîte à outils, cliquez sur le bouton Boîte à outils de la barre d'outils Standard ou sélectionnez la commande Boîte à outils du menu Affichage.

Figure 12.4
La boîte à outils contient des
contrôles Windows usuels.



Outil Sélection



L'outil Sélection permet de sélectionner un ou plusieurs contrôles sur une feuille. Il est sélectionné par défaut dans la boîte à outils. Lorsque vous sélectionnez un contrôle dans la boîte à outils, l'outil Sélection redevient actif dès que le contrôle est déposé sur la feuille. Si vous sélectionnez un contrôle dans la boîte à outils, puis décidez de ne plus le placer sur la feuille, cliquez sur l'outil Sélection.

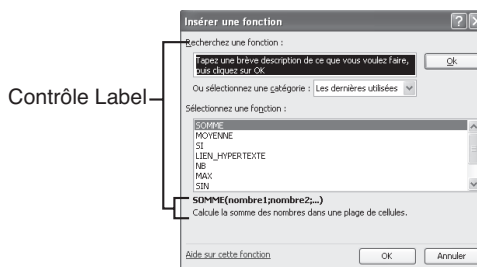
Contrôle *Label*



Le contrôle Label, ou intitulé, permet de placer un intitulé sur la feuille. Il sert généralement à placer un intitulé à côté d'un contrôle ne possédant pas cet attribut (à côté d'une zone de texte par exemple), afin de permettre à l'utilisateur d'en identifier la fonction.

Figure 12.5

Le contrôle Label permet d'attacher un libellé à un contrôle n'en possédant pas.



Contrôle *TextBox*



Le contrôle TextBox permet de placer une zone de texte sur la feuille, dans laquelle l'utilisateur pourra saisir des informations. Les propriétés d'un contrôle TextBox permettent de contrôler le nombre de caractères maximum que pourra entrer l'utilisateur, l'affichage sur plusieurs lignes du texte ou encore le type d'alignement du texte dans la zone de texte. Elles sont étudiées au chapitre suivant.

Figure 12.6

Le contrôle TextBox est l'un des contrôles les plus utilisés.

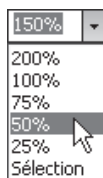
Contrôle *ComboBox*



Le contrôle ComboBox, ou zone de liste modifiable, est une zone de texte permettant à l'utilisateur de saisir une valeur manuellement ou de la sélectionner dans la liste qui se déroule lorsqu'il clique sur le bouton prévu à cet effet. Un contrôle ComboBox peut permettre à l'utilisateur de saisir une valeur ne figurant pas dans la liste, ou n'autoriser qu'une des valeurs de la liste.

Figure 12.7

Le contrôle ComboBox permet à l'utilisateur de sélectionner une valeur dans la liste ou de choisir une valeur de son choix.



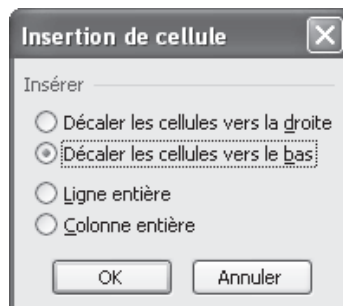
Contrôle *Frame*



Le contrôle Frame ou cadre permet de placer un cadre présentant un intitulé sur une feuille, afin d'y placer des contrôles. Il est généralement utilisé pour distinguer aisément les catégories de contrôles d'une feuille. Les cadres servent aussi à associer des contrôles `OptionButton` : il suffit de placer les boutons d'option sur un même cadre pour que ceux-ci soient associés (voir la section "Contrôle *OptionButton*", ci-après).

Figure 12.8

Un contrôle Frame (ici intitulé *Insérer*) permet de légender un ensemble de contrôles.



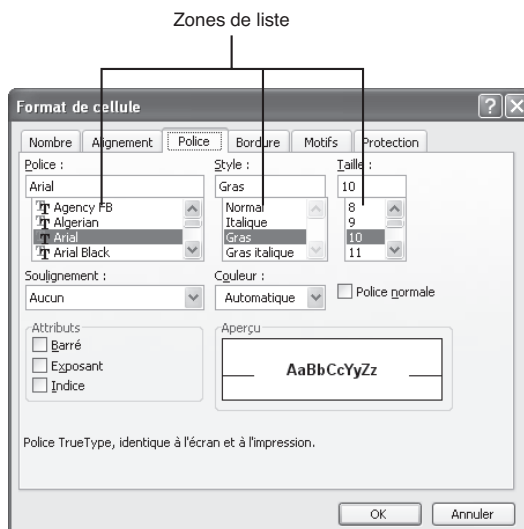
Contrôle *ListBox*



Le contrôle *ListBox*, ou zone de liste, est une zone affichant des options parmi lesquelles l'utilisateur peut opérer des sélections. Les propriétés d'un contrôle *ListBox* permettent d'autoriser la sélection d'un seul ou de plusieurs éléments de la liste. Lorsque la hauteur d'une zone de liste ne permet pas d'en afficher tous les éléments, une barre de défilement verticale lui est associée.

Figure 12.9

Une zone de texte peut être associée à une zone de liste afin d'en afficher la valeur sélectionnée.



Contrôle *CheckBox*

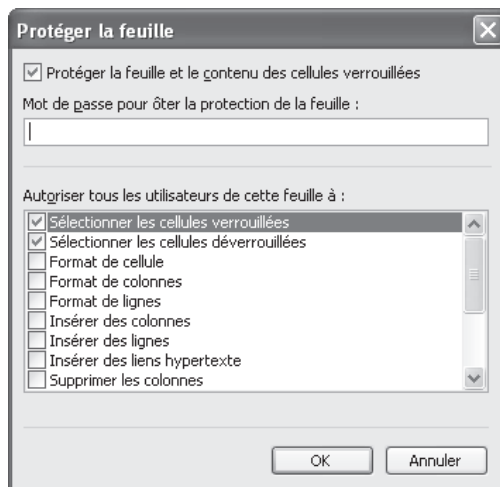


Un contrôle *CheckBox*, ou case à cocher, peut être activé (coché) ou désactivé (non coché) par l'utilisateur, afin de valider ou non une option. Les propriétés d'un contrôle *CheckBox*

permettent aussi de lui faire accepter un troisième état, Null (ni coché, ni décoché). La case à cocher apparaît alors grisée.

Figure 12.10

Les cases à cocher permettent de valider ou non une option.



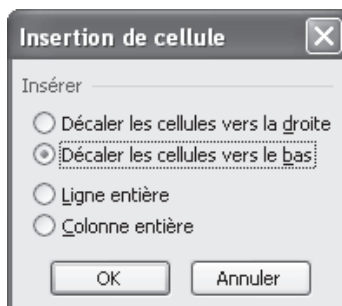
Contrôle *OptionButton*



Les contrôles `OptionButton` ou boutons d'option permettent de proposer à l'utilisateur un choix parmi plusieurs options. Lorsque plusieurs contrôles `OptionButton` sont associés, seul un d'eux peut être activé. L'activation d'un bouton d'option entraîne la désactivation des autres boutons d'option associés.

Figure 12.11

Un seul bouton d'option peut être sélectionné.



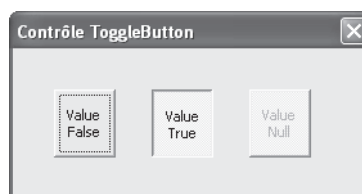
Contrôle *ToggleButton*



Le contrôle `ToggleButton`, ou bouton bascule, permet à l'utilisateur d'activer ou de désactiver une option, voire de choisir l'état Null si les propriétés du contrôle l'y autorisent. Ce contrôle offre en fait les mêmes fonctionnalités qu'une case à cocher, avec l'apparence d'un bouton de commande. Lorsqu'un bouton bascule est activé, il semble enfoncé sur la feuille ; lorsqu'il est désactivé, il est saillant ; à l'état Null, il apparaît estompé. La Figure 12.12 présente les trois états possibles pour un contrôle `ToggleButton`.

Figure 12.12

Le contrôle `ToggleButton` est assez rarement utilisé dans les interfaces de programme.



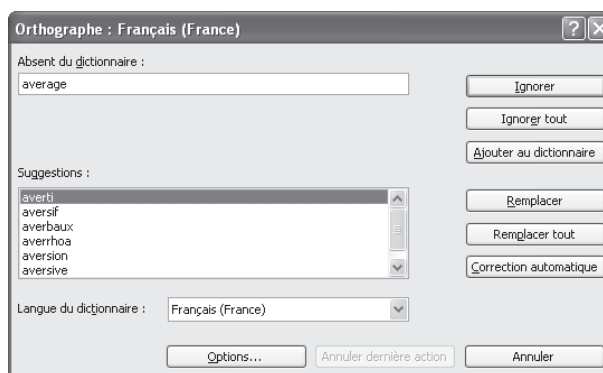
Contrôle *CommandButton*



Le contrôle `CommandButton` ou bouton de commande est un bouton associé à une action. Il peut, par exemple, servir à valider les informations entrées dans la feuille (bouton OK) afin de passer à l'étape suivante du programme, ou au contraire interrompre le programme (bouton Annuler).

Figure 12.13

Les boutons de commande figurent parmi les contrôles les plus courants.



Contrôle *TabStrip*



Le contrôle `TabStrip`, ou onglet, permet de créer une feuille dont l'affichage variera selon l'onglet sélectionné. Tout se passe comme si l'accès à plusieurs feuilles était dirigé à partir

d'une seule feuille. Les onglets permettent de résoudre des problèmes d'espace en organisant les contrôles d'une feuille par types, classés sous différents onglets. Lorsque l'utilisateur clique sur un onglet, les contrôles de la feuille sont mis à jour par le code associé au contrôle.

Notez que les contrôles ne peuvent être associés à un onglet d'un contrôle `TabStrip`, ce contrôle ne possédant pas un espace propre à chaque onglet. Les contrôles sont placés sur la feuille et leur mise à jour liée à la sélection d'un onglet ne peut être effectuée que par l'exécution de code précédemment écrit.

Contrôle *MultiPage*



Le contrôle `MultiPage` est semblable au contrôle `TabStrip`, en ce sens qu'il permet d'économiser de l'espace en associant différents affichages à une même feuille. À la différence du contrôle `TabStrip`, les pages constituant le contrôle `MultiPage` sont autonomes de la feuille. Autrement dit, lorsque vous placez des contrôles sur une page d'un contrôle `MultiPage`, ceux-ci ne s'afficheront que si la page en question est sélectionnée. Vous n'aurez donc pas à écrire de code pour faire varier l'affichage de la feuille, mais simplement à placer les contrôles voulus sur les pages constituant le contrôle.

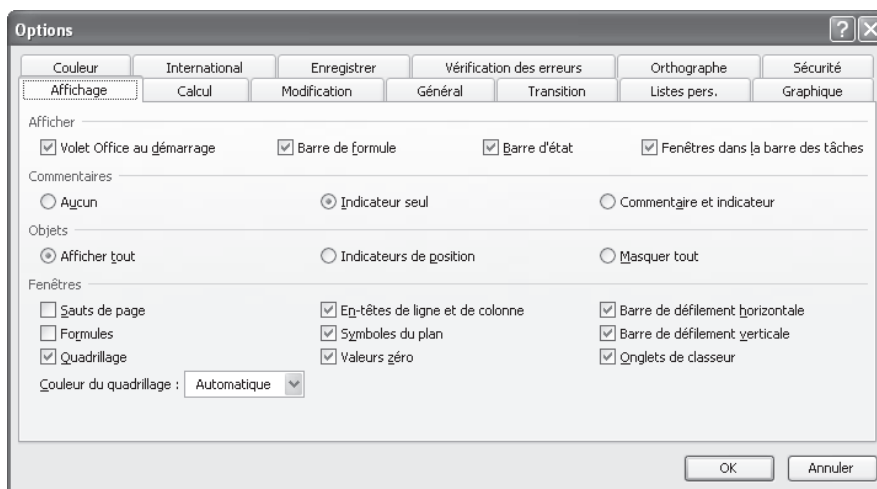


Figure 12.14

Plusieurs contrôles `MultiPage` peuvent être placés sur une feuille, chacun pouvant présenter un nombre variable d'onglets.

Contrôle *ScrollBar*

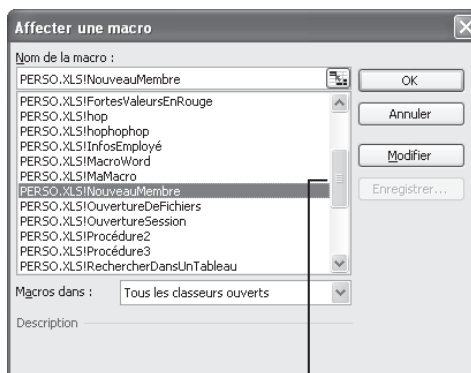


Le contrôle `ScrollBar`, ou barre de défilement, offre à l'utilisateur une zone lui permettant de se déplacer dans un environnement. Une barre de défilement peut être horizontale ou verticale. Pour déterminer la position d'une barre de défilement, redimensionnez-la en utilisant ses poignées de sélection. Le déplacement à l'aide d'une barre de défilement peut s'effectuer par un clic sur l'une des touches fléchées de la barre, un clic à l'intérieur de la barre (entre le curseur de défilement et l'une des touches fléchées), ou encore par un glisser-déplacer du curseur de défilement sur la barre.

Un contrôle `ScrollBar` est automatiquement affecté à un contrôle `ListBox` lorsque la hauteur de ce dernier ne permet pas l'affichage de tous les éléments de la liste.

Figure 12.15

Un contrôle `ScrollBar` permet à l'utilisateur de se déplacer dans une zone de texte trop petite pour afficher la totalité de son contenu.



Contrôle `ScrollBar`

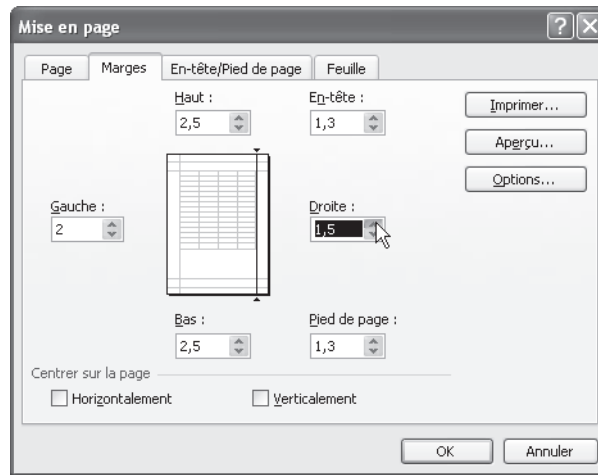
Contrôle *SpinButton*



Un contrôle `SpinButton`, ou bouton toupie, est composé de deux flèches permettant à l'utilisateur de sélectionner une valeur en cliquant sur l'une ou l'autre des flèches. Les boutons toupies sont généralement associés à une zone de texte dans laquelle s'affiche la valeur sélectionnée. Les propriétés de la zone de texte peuvent être définies pour que l'utilisateur puisse y entrer une valeur saisie au clavier ou pour que la valeur ne puisse être définie qu'à l'aide du bouton toupie.

Figure 12.16

Un contrôle TextBox est généralement associé à un contrôle SpinButton afin d'afficher la valeur choisie à l'aide de ce dernier.



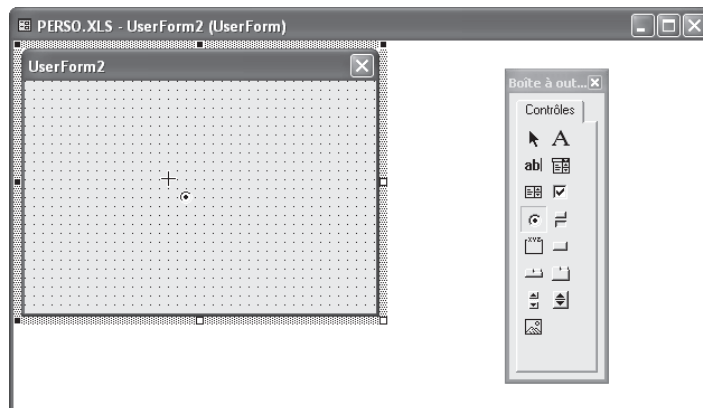
Placer des contrôles sur une feuille

Pour placer un contrôle sur une feuille :

1. Dans la boîte à outils, cliquez sur le contrôle voulu. Le contrôle prend l'apparence d'un bouton enfoncé.
2. Déplacez le pointeur au-dessus de la feuille. Le pointeur se transforme en croix et une image représentant le type de contrôle sélectionné lui est accolée (voir Figure 12.17).

Figure 12.17

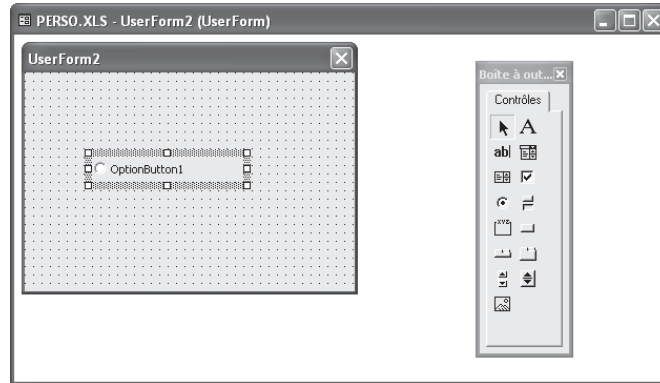
Une image accolée au pointeur indique le type de contrôle sélectionné.



- Placez la croix à l'endroit où vous souhaitez voir apparaître l'angle supérieur gauche du contrôle et cliquez. Le contrôle apparaît sur la feuille (voir Figure 12.18) et l'outil Sélection est activé dans la boîte à outils.

Figure 12.18

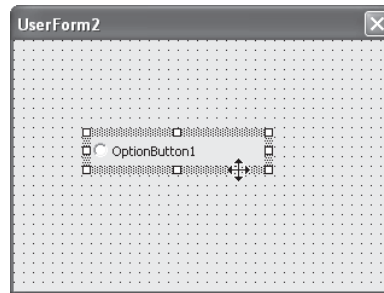
Le contrôle OptionButton déposé sur la feuille.



- Pour déplacer le contrôle, sélectionnez-le si nécessaire. Placez le curseur sur la bande grisée représentant la sélection, de façon qu'il prenne la même forme qu'à la Figure 12.19, et effectuez un glisser-déplacer jusqu'à l'emplacement voulu.

Figure 12.19

Cette forme de pointeur indique que vous pouvez déplacer le contrôle.



- Pour redimensionner le contrôle, sélectionnez-le si nécessaire, puis utilisez les poignées de redimensionnement situées autour du cadre de sélection (voir Figure 12.20).

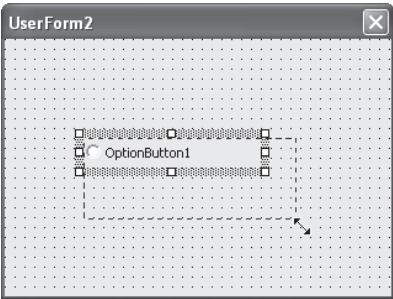
Conseil

Dès que vous avez placé un contrôle sur une feuille, définissez sa propriété Name dans la fenêtre Propriétés. La propriété Name d'un contrôle correspond au nom qui sera utilisé dans le code pour faire référence à ce contrôle.

Astuce

Pour accéder à la rubrique d'aide Visual Basic d'un contrôle, sélectionnez ce contrôle sur la feuille et tapez sur la touche F1.

Figure 12.20
Utilisez les poignées de redimensionnement pour ajuster la taille du contrôle.



Ce chapitre traite du développement de feuilles VBA. De la même façon, des contrôles ActiveX peuvent être placés sur une feuille de calcul Excel. Les procédures d'exploitation de ces contrôles sont les mêmes que pour une feuille UserForm développée dans Visual Basic Editor. L'objet correspondant à la feuille UserForm et dans lequel sont stockées les procédures événementielles des contrôles est l'objet Feuille portant le nom de la feuille, situé dans le module Objets Microsoft Excel.

Pour placer des contrôles ActiveX sur une feuille de calcul plutôt que sur une feuille UserForm, affichez la barre d'outils Visual Basic dans Excel et cliquez sur le bouton Création. Cliquez ensuite sur le bouton Boîte à outils Contrôles, afin d'afficher les contrôles disponibles. Pour quitter le mode Création, cliquez à nouveau sur le bouton Création.

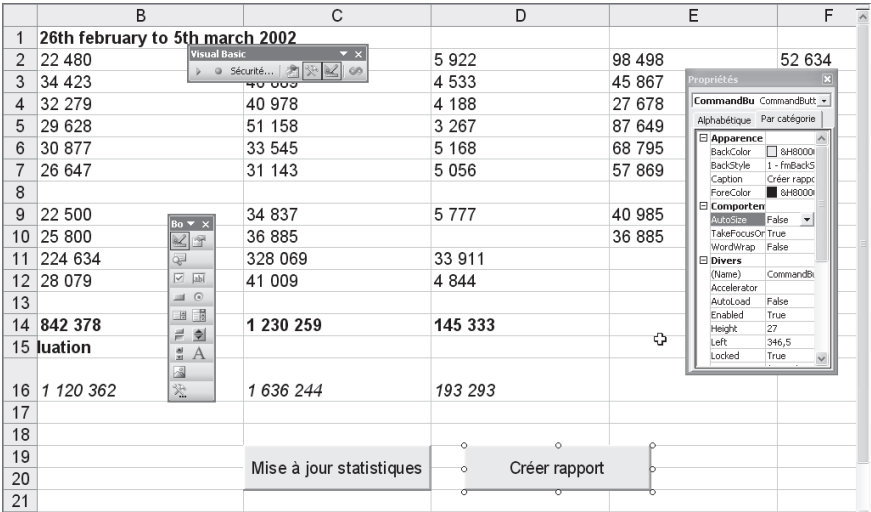


Figure 12.21
Vous pouvez aussi placer des contrôles sur une feuille de calcul Excel.

Copier-coller des contrôles

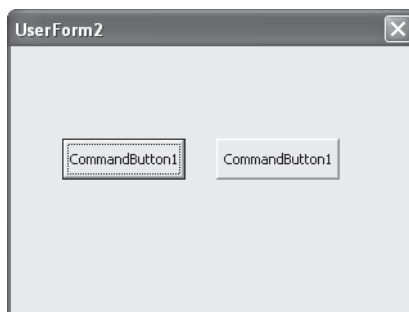
Si vous souhaitez placer plusieurs contrôles identiques sur une feuille, vous pouvez répéter la procédure indiquée précédemment. Cependant, il est plus rapide de placer un contrôle sur la feuille, de le dimensionner à votre convenance et d'en déterminer les propriétés (voir section suivante), puis de le copier et de le coller sur la feuille. Le contrôle ainsi dupliqué héritera des propriétés du contrôle stocké dans le Presse-papiers, et vous éviterez ainsi d'avoir à les redéfinir.

Pour copier et coller un contrôle :

1. Sélectionnez le contrôle sur la feuille, puis copiez-le afin de le placer dans le Presse-papiers. Vous pouvez pour cela utiliser le bouton Copier de la barre d'outils, sélectionner la commande Copier du menu Édition ou du menu contextuel (clic droit sur le contrôle), ou encore le raccourci clavier Ctrl+C.
2. Collez ensuite le contenu du Presse-papiers (bouton Coller, commande Coller du menu Édition ou du menu contextuel, ou encore le raccourci clavier Ctrl+V).
3. Déplacez si nécessaire le nouveau contrôle sur la feuille.
4. Ouvrez ensuite la fenêtre Propriétés du contrôle en appuyant sur F4 ou en faisant un clic droit sur le contrôle et en choisissant la commande Propriétés du menu contextuel. Définissez sa propriété Name.

Figure 12.22

La copie du contrôle hérite des propriétés du contrôle source.



Astuce

Pour copier-coller un contrôle, vous pouvez aussi cliquer du bouton droit sur le contrôle et, tout en maintenant le bouton de la souris enfoncé, le faire glisser à l'endroit où vous souhaitez en placer une copie. Relâchez ensuite le bouton de la souris et, dans le menu contextuel qui s'affiche, sélectionnez Copier ici.

Astuce

Pour dupliquer plusieurs contrôles en une seule opération, sélectionnez les contrôles voulus sur la feuille.

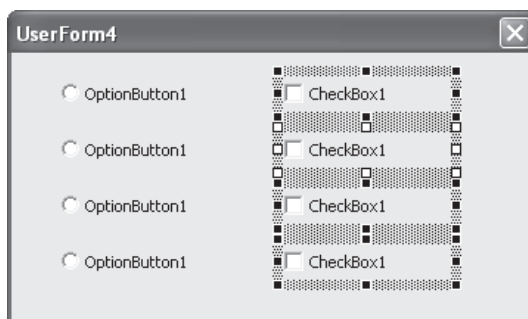
Sélectionner plusieurs contrôles

Pour sélectionner plusieurs contrôles sur une feuille, utilisez les touches Maj et Ctrl :

- Pour sélectionner des contrôles contigus, sélectionnez le premier contrôle. Appuyez ensuite sur la touche Maj et, tout en la maintenant enfoncée, sélectionnez le dernier contrôle.
- Pour sélectionner des contrôles non contigus, sélectionnez le premier contrôle. Appuyez ensuite sur la touche Ctrl et, tout en la maintenant enfoncée, sélectionnez successivement les contrôles voulus.

Figure 12.23

Pour sélectionner plusieurs contrôles sur une feuille, utilisez les touches Maj et Ctrl.



Info

Lorsque vous sélectionnez plusieurs contrôles, les poignées de sélection de l'un d'eux apparaissent en blanc (celles des autres contrôles sont noires). Il s'agit du contrôle actif. Cette notion est particulièrement utile pour mettre en forme vos feuilles. Vous pouvez ainsi appliquer certaines propriétés du contrôle actif – telles que sa taille ou sa position – aux autres contrôles sélectionnés.

Supprimer des contrôles

Pour effacer des contrôles d'une feuille, sélectionnez les contrôles voulus et tapez sur la touche Suppr pour les supprimer sans les stocker dans le Presse-papiers. Si vous souhaitez coller les contrôles dans une autre feuille, tapez Ctrl+X afin de placer les contrôles supprimés dans le Presse-papiers. Activez ensuite la feuille devant recevoir les contrôles et tapez Ctrl+V afin de coller ces derniers dans la feuille active.

Mise en forme des contrôles

Le simple déplacement des contrôles sur une feuille à l'aide de la souris n'est pas assez précis pour atteindre une mise en forme parfaite des contrôles. Un certain nombre d'outils sont à votre disposition pour peaufiner la mise en forme des contrôles sur une feuille. Vous obtiendrez ainsi des contrôles parfaitement alignés et de dimensions proportionnées. La mise en forme des contrôles sur la feuille peut se faire à l'aide de la grille, des commandes du menu Format ou de la barre d'outils UserForm (voir Figure 12.24).

Figure 12.24

La barre d'outils UserForm.



La grille

La grille désigne les points quadrillant une feuille dans une fenêtre UserForm. Ces points n'apparaissent qu'en phase de conception et ont pour fonction de faciliter la mise en place des contrôles sur la feuille.

Lorsque l'alignement des contrôles sur la grille est activé, l'angle supérieur gauche des contrôles est automatiquement placé sur le point de la grille le plus proche et les contrôles ne peuvent être placés ou déplacés que sur l'un des points de la grille. Les contrôles acceptent alors autant de positions qu'il existe de points de quadrillage sur la feuille. Pour activer l'alignement des contrôles sur la grille :

1. Sélectionnez la commande Options du menu Outils de Visual Basic Editor, et activez la page Général de la boîte de dialogue qui s'affiche. La zone Paramètres de grille de la feuille (voir Figure 12.25) permet de définir les options d'utilisation de la grille.
2. Cochez les cases Afficher la grille afin de faire apparaître les points de quadrillage sur la feuille, et déterminez la précision de la grille en spécifiant des valeurs de largeur et de hauteur.
3. Cochez la case Aligner les contrôles sur la grille, puis cliquez sur OK.



La commande Ajuster à la grille du menu Format ajuste la hauteur et la largeur des contrôles sélectionnés aux lignes de grille les plus proches.

Figure 12.25
*Paramétrez l'utilisation
de la grille.*

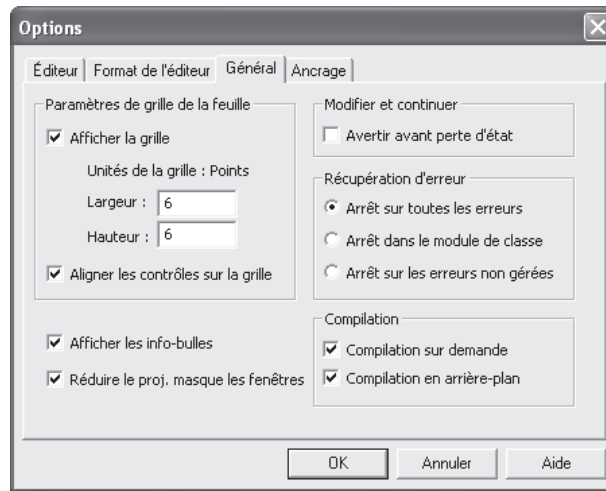
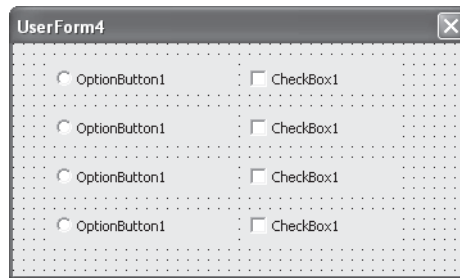


Figure 12.26
*L'alignement des contrôles
sur la grille rend aisé
l'alignement des contrôles les
uns par rapport aux autres.*



Aligner les contrôles

Pour aligner des contrôles sur une feuille :

1. Sélectionnez les contrôles à aligner en prenant soin de faire du contrôle qui servira de référence pour l'alignement le contrôle actif (celui dont les poignées de sélection apparaissent en blanc).

Astuce

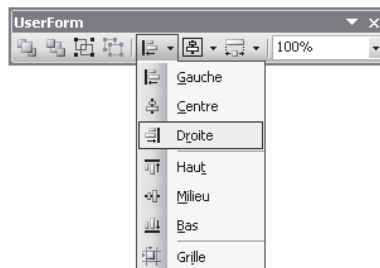
Pour modifier le contrôle actif d'une sélection, maintenez la touche **Ctrl** enfoncée et cliquez à deux reprises sur le contrôle que vous souhaitez rendre actif. Le premier clic désélectionnera le contrôle, le second le sélectionnera à nouveau en le rendant actif.

2. Choisissez la commande Aligner du menu Format ou cliquez sur la flèche de déroulement du bouton Aligner de la barre d'outils UserForm (voir Figure 12.27).
3. Choisissez l'alignement voulu parmi les sept types d'alignement proposés.

Les contrôles sélectionnés s'alignent sur le contrôle actif.

Figure 12.27

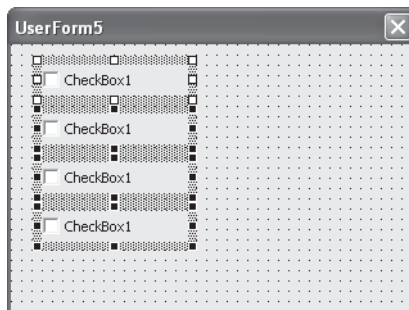
Sélectionnez un type d'alignement.



Un mauvais choix peut entraîner la juxtaposition des contrôles. Cliquez alors sur le bouton Annuler de la barre d'outils Standard ou tapez le raccourci Ctrl+Z pour ramener les contrôles à leur position précédente.

Figure 12.28

L'alignement des contrôles se fait sur le contrôle actif de la sélection.



Uniformiser la taille des contrôles

Pour donner la même taille à différents contrôles d'une feuille :

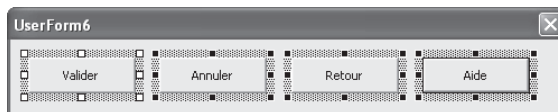
1. Sélectionnez les contrôles dont vous souhaitez uniformiser la taille, en prenant soin de faire du contrôle qui servira de référence le contrôle actif.

2. Choisissez la commande Uniformiser la taille du menu Format ou cliquez sur la flèche de déroulement du bouton Égaliser de la barre d'outils UserForm.
3. Choisissez d'uniformiser la hauteur des contrôles, leur largeur ou encore les deux.

Les contrôles sélectionnés prennent tous la taille du contrôle actif.

Figure 12.29

Une même largeur et une même hauteur ont été affectées aux contrôles sélectionnés.



Vous pouvez adapter la taille d'un contrôle à son contenu. Sélectionnez pour cela le contrôle et choisissez la commande Ajuster la taille du menu Format.

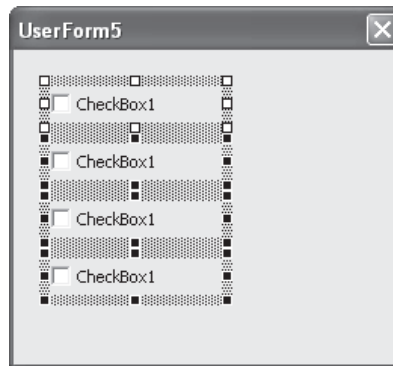
Uniformiser l'espace entre les contrôles

Pour uniformiser l'espace séparant les contrôles d'une feuille :

1. Sélectionnez les contrôles dont vous souhaitez uniformiser l'espace de séparation, en prenant soin de faire du contrôle qui servira de référence le contrôle actif.
2. Choisissez la commande Espacement horizontal ou Espacement vertical du menu Format, en fonction des positions réciproques des contrôles.
3. Choisissez l'une des commandes du menu qui s'affiche :
 - **Égaliser.** L'espacement entre les contrôles sélectionnés sera le même.
 - **Augmenter.** L'espacement entre les contrôles sélectionnés augmentera d'une unité de grille, soit d'une distance égale à l'espace séparant deux points adjacents de la grille.
 - **Diminuer.** L'espacement entre les contrôles sélectionnés diminuera d'une unité de grille, soit d'une distance égale à l'espace séparant deux points adjacents de la grille.
 - **Supprimer.** L'espace séparant les contrôles sélectionnés sera supprimé. Pour autant, les contrôles ne seront pas juxtaposés, mais accolés.

Figure 12.30

Un même espacement vertical a été appliqué aux contrôles sélectionnés.



Vous pouvez adapter la taille d'un contrôle à son contenu. Sélectionnez pour cela le contrôle et choisissez la commande Ajuster la taille du menu Format.

Centrer les contrôles

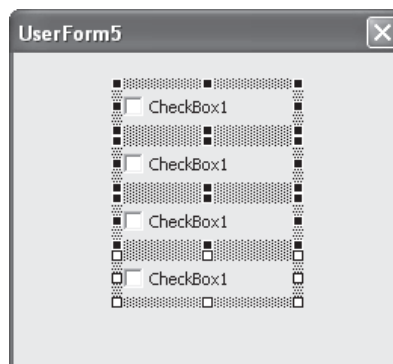
Pour centrer des contrôles sur une feuille :

1. Sélectionnez les contrôles à centrer, en prenant soin de faire du contrôle qui servira de référence le contrôle actif.
2. Choisissez la commande Centrer sur la feuille du menu Format ou cliquez sur la flèche de déroulement du bouton Centrer de la barre d'outils UserForm.
3. Choisissez un centrage horizontal ou vertical, en fonction de la position des contrôles sur la feuille.

Les contrôles sélectionnés sont centrés sur la feuille.

Figure 12.31

Les contrôles ont été centrés horizontalement sur la feuille.



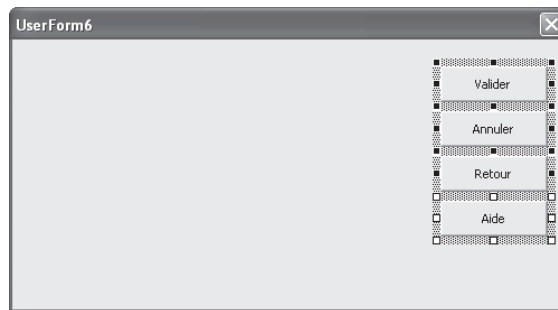
Réorganiser les boutons

Pour placer les boutons dans la partie inférieure ou droite d'une feuille :

1. Sélectionnez les contrôles `CommandButton` à déplacer.
2. Choisissez la commande Réorganiser les boutons du menu `Format`.
3. Choisissez l'une des options de réorganisation :
 - **En bas.** Les boutons seront placés en bas et à gauche de la feuille et séparés par un même espace.
 - **À droite.** Les boutons seront placés à droite et en haut de la feuille et séparés par un même espace.

Figure 12.32

Les boutons ont été placés à droite de la feuille.



Info

Si des contrôles autres que des boutons de commande sont sélectionnés, ils ne seront pas affectés par la commande Réorganiser les boutons. Si des contrôles occupent déjà l'espace vers lequel sont envoyés les boutons, ces derniers les recouvriront.

Grouper ou séparer des contrôles

Lorsque des contrôles sont groupés, un clic sur l'un des contrôles du groupe sélectionne le groupe entier, un cadre permet de distinguer les contrôles du groupe (voir Figure 12.33). Un second clic sur l'un des contrôles du groupe sélectionne ce contrôle. Vous pouvez ainsi agir sur ce contrôle sans affecter les autres contrôles du groupe.

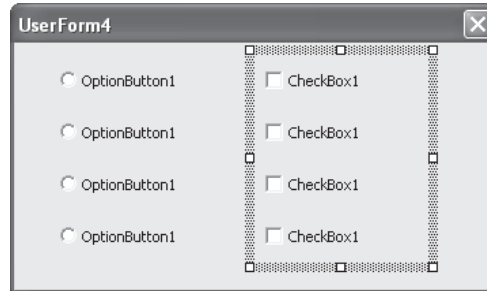
Grouper les contrôles présente plusieurs avantages :

- Les contrôles peuvent être sélectionnés en un seul clic. Vous pouvez ainsi les déplacer simultanément en déplaçant le cadre de sélection du groupe.

- Vous pouvez redimensionner les contrôles simultanément, en redimensionnant simplement le cadre de sélection du groupe. Les modifications de taille du cadre seront appliquées à l'identique à chacun des contrôles du groupe.
- Lorsqu'un groupe de contrôles est sélectionné, la fenêtre Propriétés affiche les propriétés communes aux contrôles du groupe. La redéfinition de la valeur d'une propriété affecte alors tous les contrôles.

Figure 12.33

Un clic sur un contrôle appartenant à un groupe sélectionne le groupe entier.



On ne peut sélectionner qu'un contrôle à la fois à l'intérieur d'un groupe. Vous ne pouvez donc pas accéder à l'essentiel des commandes de mise en forme des contrôles. Déterminez donc la mise en forme des contrôles avant de les grouper.

Pour grouper des contrôles :

1. Sélectionnez les contrôles que vous souhaitez grouper.
2. Choisissez la commande Grouper du menu Format ou cliquez sur le bouton Grouper de la barre d'outils UserForm.

Pour dissocier les contrôles d'un groupe :

1. Sélectionnez le groupe.
2. Choisissez la commande Séparer du menu Format ou cliquez sur le bouton Séparer de la barre d'outils UserForm.

Personnaliser la boîte à outils

Les contrôles présentés précédemment sont les contrôles standard, disponibles par défaut dans la boîte à outils. Vous pouvez cependant personnaliser la boîte à outils, en y ajoutant des contrôles, que vous placerez au côté des contrôles déjà disponibles ou sur une nouvelle page, ou en en supprimant.

Ajouter/supprimer un contrôle

Dans cet exemple, nous ajouterons le contrôle Calendar aux contrôles de la boîte à outils. Ce contrôle ActiveX n'étant plus livré avec Office 2010, vous commencerez par l'installer sur votre machine. Procédez comme suit :

1. Quittez les applications Office. Téléchargez l'archive du contrôle à l'adresse suivante : www.edito.biz/docs/vba/MSCAL.OCX.
2. Placez ce fichier dans le dossier approprié, à savoir C:/Windows/System32 pour Windows XP, Windows Vista et Windows 7.
3. Enregistrez maintenant le contrôle activeX dans la base de registres : lancez la commande Exécuter (Programmes > Accessoires > Exécuter), puis saisissez le texte suivant **regsvr32 mscal.ocx**. Cliquez sur le bouton OK.
4. Relancez Excel.

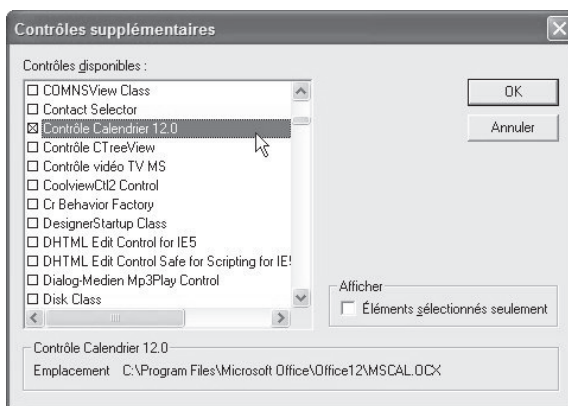
Pour ajouter le contrôle Calendrier à la boîte à outils, procédez ainsi :

1. Affichez la boîte à outils. Cliquez du bouton droit sur celle-ci et sélectionnez la commande Contrôles supplémentaires du menu contextuel ou choisissez la commande Contrôles supplémentaires du menu Outils.

La boîte de dialogue Contrôles supplémentaires s'affiche (voir Figure 12.34). La zone de liste Contrôles disponibles recense les contrôles pouvant être exploités et placés sur la boîte à outils.

Figure 12.34

La boîte de dialogue Contrôles supplémentaires.



2. Pour affecter un nouveau contrôle à la boîte à outils, cochez sa case. Pour supprimer un contrôle disponible sur la boîte à outils, décochez sa case. Ici, cochez la case intitulée Contrôle Calendrier 11.0.

3. Cliquez sur OK. Le contrôle Calendar est maintenant disponible dans la boîte à outils (voir Figure 12.35).

Figure 12.35

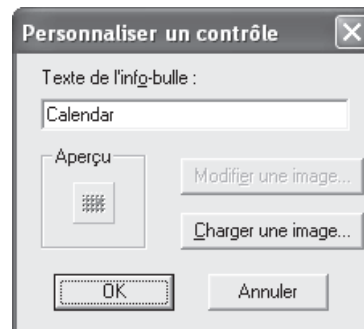
Le contrôle Calendar a été ajouté à la boîte à outils.



4. Pour personnaliser le contrôle dans la boîte à outils, cliquez du bouton droit sur celui-ci et sélectionnez *Personnaliser contrôle*. La boîte de dialogue *Personnaliser un contrôle* s'affiche (voir Figure 12.36).

Figure 12.36

La boîte de dialogue Personnaliser un contrôle.



5. Vous pouvez :
 - Modifier le texte de l'info-bulle du contrôle en tapant le texte voulu dans la zone de texte Texte de l'info-bulle.
 - Modifier l'image représentant le contrôle sur la boîte à outils, en chargeant un fichier image (bouton Charger une image) ou en dessinant vous-même l'image qui sera affichée dans la boîte à outils (Éditer une image). Cette dernière possibilité

ouvrira la fenêtre Éditer une image. L'utilisation de l'éditeur d'image est présentée au Chapitre 11.

Astuce

Pour supprimer des contrôles à partir de la boîte de dialogues Contrôles supplémentaires, cochez la case Afficher éléments sélectionnés seulement. Seuls les contrôles disponibles sur la boîte à outils seront affichés dans la zone de liste Contrôles disponibles. Décochez alors les contrôles que vous souhaitez supprimer.

Vous pouvez aussi supprimer un contrôle à partir de la boîte à outils. Cliquez du bouton droit sur le contrôle à supprimer et sélectionnez la commande Supprimer contrôle.

La Figure 12.37 présente une feuille sur laquelle un contrôle Calendar a été placé.

Figure 12.37

Le contrôle Calendar permet à l'utilisateur de sélectionner une date sur un calendrier.

Dates de remise et de parution

Indiquez la date à valider et sélectionnez-la sur le calendrier.

Date de remise : 09/12/2003

Date de parution : 27/12/2003

Date à valider : Date de parution

December 2003 December 2003

lun.	mar.	mer.	jeu.	ven.	sam.	dim.
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Annuler Retour Terminer

Ajouter/supprimer une page

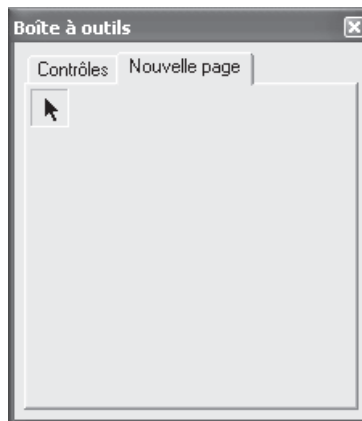
Si vous exploitez de nombreux contrôles, il sera judicieux de regrouper les contrôles par catégories que vous placerez sur des pages distinctes de la boîte à outils.

Pour ajouter une page à la boîte à outils :

1. Cliquez sur l'onglet Contrôles de la boîte à outils et sélectionnez la commande Nouvelle page. Une nouvelle page apparaît sur la boîte à outils (voir Figure 12.38). Par défaut, le libellé de son onglet est Nouvelle page, et l'outil Sélection y est disponible.

Figure 12.38

Une nouvelle page ajoutée à la boîte à outils.

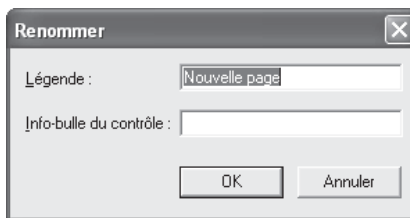


L'outil Sélection est toujours présent sur les pages de la boîte à outils et ne peut être supprimé.

2. Cliquez du bouton droit sur l'onglet de la nouvelle page et choisissez la commande Renommer. La boîte de dialogue Renommer s'affiche (voir Figure 12.39). Indiquez le nom de la page qui apparaîtra sur son onglet et, éventuellement, le texte d'info-bulle. Cliquez sur OK.

Figure 12.39

La boîte de dialogue Renommer.

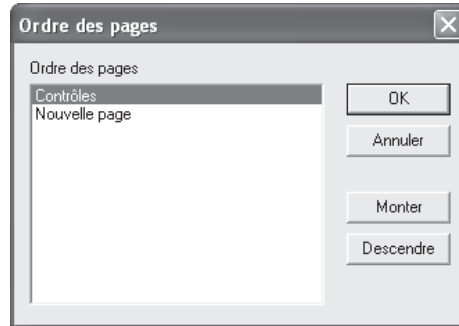


3. Pour ajouter ou supprimer des contrôles sur la nouvelle page, activez-la, puis procédez comme indiqué dans la section précédente.
4. Si vous souhaitez modifier l'emplacement de l'onglet de la page sur la boîte à outils, cliquez sur celui-ci et sélectionnez la commande Déplacer. Utilisez les boutons Vers

le haut et Vers le bas de la boîte de dialogue Ordre des pages (voir Figure 12.40) pour modifier l'ordre des onglets sur la boîte à outils.

Figure 12.40

L'emplacement des onglets de la boîte à outils se détermine dans cette boîte de dialogue.



Pour supprimer une page, cliquez du bouton droit sur son onglet et choisissez la commande Supprimer la page.

Afficher/masquer une feuille

Une feuille ne peut s'exécuter d'elle-même. Son affichage doit être commandé à partir d'une procédure d'un module de code du projet. On utilise pour cela la méthode Show, selon la syntaxe suivante :

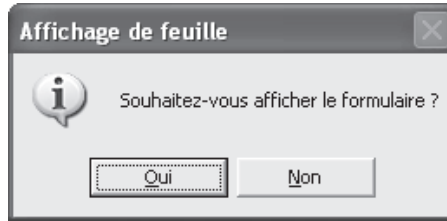
NomFeuille.Show

où NomFeuille est le nom de la feuille à afficher, tel qu'il apparaît dans l'Explorateur de projet. La procédure suivante affiche une boîte de dialogue invitant l'utilisateur à indiquer s'il souhaite que l'interface soit affichée. Si celui-ci répond positivement, la feuille nommée MaFeuille s'affiche. Dans le cas contraire, le programme se poursuit avec la procédure SuiteDuProgramme.

```
Sub AffichageFeuille()  
    Dim RepAffichage As Integer  
    RepAffichage = MsgBox("Souhaitez-vous afficher le formulaire", _  
        vbYesNo + vbQuestion, "Affichage de feuille")  
    If RepAffichage = vbYes Then  
        MaFeuille.Show  
    Else  
        Call SuiteDuProgramme  
    End If  
End Sub
```

Figure 12.41

Si l'utilisateur clique sur Oui, la méthode Show sera appliquée à la feuille afin de l'afficher.



Pour masquer une feuille, on lui applique la méthode Hide, selon la syntaxe suivante :

```
NomFeuille.Hide
```

Cette instruction est généralement placée dans la procédure événementielle affectée au bouton de validation de la feuille. Les valeurs entrées par l'utilisateur dans la feuille sont alors passées à une autre procédure qui les exploitera, et la feuille est masquée. Dans l'exemple suivant, lorsque l'utilisateur clique sur le bouton cmdOK de la feuille nommée MaFeuille, celle-ci est masquée, et la procédure SuiteDuProgramme est appelée.

```
Private Sub cmdOK_Click()  
    MaFeuille.Hide  
    Call SuiteDuProgramme(arg1, arg2, ..., argN)  
End Sub
```

Ici, *arg1*, *arg2*, ..., *argN* représentent les arguments passés à la procédure SuiteDuProgramme.

Astuce

La propriété Me renvoie la feuille active. Elle peut être utilisée dans n'importe quelle procédure événementielle attachée à une feuille pour manipuler celle-ci. Si la feuille que vous souhaitez masquer est la feuille active (l'objet conteneur du contrôle ayant reçu l'événement), utilisez le mot clé Me à la place de la propriété Name de la feuille (Me.Hide).

Info

L'affectation de procédures événementielles aux contrôles est traitée aux Chapitres 13 et 14.



13

Exploiter les propriétés des contrôles ActiveX

Au sommaire de ce chapitre

- Propriété *Name*
- Apparence
- Comportement
- Défilement
- Divers
- Emplacement
- Image
- Police

Au cours des chapitres précédents, vous avez découvert les différents contrôles disponibles et les outils destinés à leur mise en forme. Vous savez donc créer des formulaires à l'apparence professionnelle, mais ce ne sont pour l'instant que des feuilles sans vie. Ce chapitre et le suivant vous montrent comment tirer parti des feuilles VBA et des contrôles ActiveX en exploitant les propriétés des contrôles et en associant du code aux événements les affectant.

En tant qu'objet, les feuilles et les contrôles possèdent un certain nombre de propriétés qui en déterminent l'apparence et le comportement. Il est fortement recommandé de déterminer la valeur de la propriété Name des contrôles au fur et à mesure que vous les placez sur la feuille.

Pour déterminer les propriétés d'une feuille ou d'un contrôle, sélectionnez l'objet en question et tapez sur la touche F4 afin d'en ouvrir la fenêtre Propriétés, ou cliquez du bouton droit sur le contrôle et sélectionnez la commande Propriétés du menu contextuel. Affectez ensuite les valeurs voulues aux propriétés de l'objet.

Pour un rappel sur l'utilisation de la fenêtre Propriétés, reportez-vous à la section "La fenêtre Propriétés" du Chapitre 4.

Astuce

Pour modifier simultanément la valeur d'une propriété de plusieurs contrôles, sélectionnez ces contrôles sur la feuille. La fenêtre Propriétés affiche alors les propriétés communes aux contrôles sélectionnés. Les propriétés définies le seront pour tous les contrôles sélectionnés.

Il existe évidemment des propriétés spécifiques pour chaque contrôle et d'autres communes à presque tous les contrôles. Les sections qui suivent vous présentent les propriétés essentielles, classées par catégories.



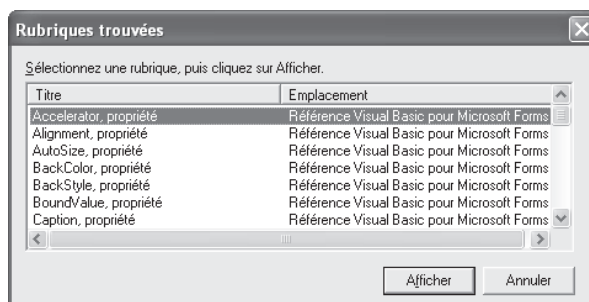
Dans la réalité, vous n'utiliserez que très peu de ces propriétés, car, pour nombre d'entre elles, la valeur par défaut au moment où le bouton est placé sur la feuille est satisfaisante. Les propriétés que vous devez connaître et que vous serez amené à utiliser présentent une icône en marge, identique à celle qui se trouve en marge de ce paragraphe. La section "Exploiter les propriétés" présente ensuite les contrôles un à un et en décrit l'utilisation et les spécificités.

Astuce

Pour afficher la rubrique d'aide d'une propriété, sélectionnez le nom de la propriété dans la fenêtre Propriétés et tapez sur la touche F1. Vous pouvez aussi sélectionner le contrôle sur la feuille, puis tapez sur F1 pour ouvrir la rubrique d'aide du contrôle. Le lien Propriétés situé dans le haut de la fenêtre ouvre une boîte de dialogue Rubriques trouvées, listant les propriétés du contrôle (voir Figure 13.1). Sélectionnez la propriété dont vous souhaitez consulter la rubrique d'aide et cliquez sur Afficher.

Figure 13.1

Accédez en un clin d'œil aux rubriques d'aide des propriétés d'un contrôle.



Propriété *Name*



La propriété *Name* est exploitée par le programme, et reste invisible à l'utilisateur final. Elle correspond au nom de l'objet et est de type *String* (chaîne de caractères). Ce nom est utilisé pour faire référence à l'objet dans le code, par exemple pour interroger ou modifier l'une de ses propriétés. Un même nom ne peut être utilisé pour plusieurs objets d'une même feuille.

Lorsque vous placez un contrôle sur une feuille, sa propriété *Name* est définie par défaut en une chaîne composée du nom de la classe de l'objet suivi d'un nombre entier. Par exemple, lorsque vous placez un contrôle *OptionButton* (bouton d'option) sur une feuille, sa propriété *Name* est définie à *OptionButton1*, *OptionButton2* s'il existe déjà un contrôle *OptionButton1*, etc.



Affectez des noms représentatifs aux contrôles de votre feuille. Il vous sera ainsi facile de savoir à quel contrôle fait référence une instruction dans le code.

L'instruction *If...End If* suivante interroge la propriété *Value* d'une zone de texte (le texte saisi par l'utilisateur) dont la propriété *Name* est définie à *txtMotDePasse*. Si la zone de texte ne contient aucune information, un message, affiché à l'aide de l'instruction *MsgBox*, demande à l'utilisateur d'entrer un mot de passe.

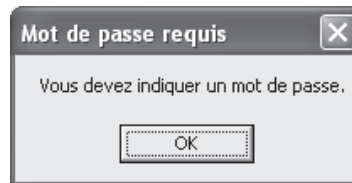
```
If txtMotDePasse.Value = "" Then
    MsgBox "Vous devez indiquer un mot de passe.", vbOKOnly, "Mot de passe requis"
End If
```


Astuce

Si une zone de texte est destinée à recevoir un mot de passe, affectez un caractère tel que l'astérisque à sa propriété `PasswordChar`, afin de masquer le mot de passe saisi. Le caractère affecté à la propriété `PasswordChar` sera affiché à la place des caractères saisis par l'utilisateur. Vous devrez aussi protéger votre projet par un mot de passe, afin que l'utilisateur ne puisse accéder au code de la procédure pour y lire le mot de passe. Pour protéger par mot de passe un projet VBA, cliquez du bouton droit sur le projet dans la fenêtre Propriétés et choisissez la commande Propriétés de NomProjet. Activez l'onglet Protection et cochez la case Verrouiller le projet pour l'affichage. Saisissez ensuite un mot de passe dans la zone de texte appropriée. La protection par mot de passe prendra effet lors du prochain lancement d'Excel.

Figure 13.2

Pour faire référence à un contrôle dans le code, utilisez sa propriété `Name`.



Apparence

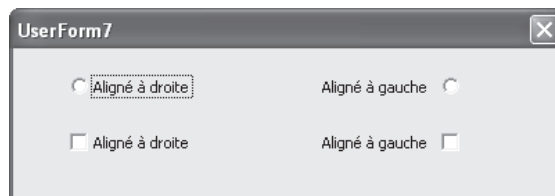
Alignment

Cette propriété détermine la position d'un contrôle `CheckBox` ou `OptionButton` par rapport à sa légende (propriété `Caption`). Elle prend pour valeur l'une des constantes `fmalignment` :

- **`fmalignmentright`** (par défaut). Place le libellé à droite du contrôle.
- **`fmalignmentleft`**. Place le libellé à gauche du contrôle.

Figure 13.3

La légende d'un contrôle `CheckBox` ou `RadioButton` peut être alignée à droite ou à gauche du contrôle.



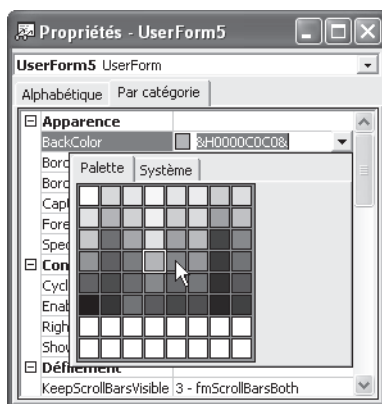
BackColor

Cette propriété détermine la couleur de fond du contrôle. Elle prend pour valeur une variable de type Long représentant une couleur. Lorsque vous modifiez la valeur de cette propriété dans la fenêtre Propriétés, une palette s'affiche (voir Figure 13.4). Vous pouvez sélectionner une couleur sur l'onglet Palette ou choisir d'appliquer une couleur système sur l'onglet Système.

Si la propriété BackStyle est définie sur fmBackStyleTransparent, cette propriété est sans effet sur l'apparence du contrôle.

Figure 13.4

Sélectionnez une couleur de fond pour le contrôle dans cette palette.



BackStyle

Cette propriété détermine le style de fond du contrôle. Elle prend pour valeur l'une des constantes fmBackStyle :

- **fmBackStyleOpaque** (par défaut). Détermine un fond opaque. Les éventuels objets placés à l'arrière-plan de l'objet sont invisibles.
- **fmBackStyleTransparent**. Détermine un fond transparent. Les éventuels objets placés à l'arrière-plan de l'objet sont visibles.

BorderColor

Cette propriété détermine la couleur du contour du contrôle et peut prendre les mêmes valeurs que la propriété BackColor.

Si la propriété BorderStyle est définie sur fmBorderStyleNone, cette propriété est sans effet sur l'apparence du contrôle.

BorderStyle

Cette propriété détermine l'affichage ou non d'une bordure pour le contrôle. Elle prend pour valeur l'une des constantes `fmBorderStyle` :

- **`fmBorderStyleNone`**. Indique qu'aucune bordure ne s'affiche pour le contrôle.
- **`fmBorderStyleSingle`**. Indique qu'une bordure s'affiche.

Caption

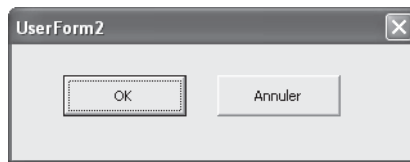


La propriété `Caption` détermine le texte descriptif d'un contrôle et accepte une valeur de type `String` (chaîne de caractères). L'emplacement de ce libellé varie selon le contrôle. Par exemple, le libellé d'un bouton de commande (`CommandButton`) est centré sur le bouton, tandis que le libellé d'une case à cocher (`CheckBox`) ou d'un bouton d'option (`OptionButton`) peut être aligné à droite (par défaut) ou à gauche du contrôle, selon la valeur de sa propriété `Alignment`.

La Figure 13.5 présente deux boutons de commande dont les propriétés `Caption` ont été définies à `OK` et à `Annuler`.

Figure 13.5

La propriété `Caption` sert à affecter une légende à un contrôle.



Il peut être utile de faire varier la propriété `Caption` d'un contrôle en fonction des événements utilisateur. Dans l'exemple suivant, lorsque l'utilisateur clique sur le bouton de commande dont la propriété `Name` est définie à `"JoueurSuivant"`, la propriété `Caption` du contrôle `MonLibellé` bascule entre `"Joueur 1"` et `"Joueur 2"`.

```
Private Sub JoueurSuivant_Click()
    Call AutreProcédure
    If MonLibellé.Caption = "Joueur 1" Then
        MonLibellé.Caption = "Joueur 2"
    Else
        MonLibellé.Caption = "Joueur 1"
    End If
End Sub

Sub AutreProcédure()
    Instructions
End Sub
```

La première ligne est une instruction de procédure d'événement (traitées au chapitre suivant) qui se déclenche lorsque l'utilisateur clique sur le bouton dont la propriété Name est "JoueurSuivant". L'instruction Call appelle la procédure AutreProcédure dont les *Instructions* s'exécutent. La procédure appelante reprend ensuite la main. Enfin, l'instruction If...End If modifie la propriété Caption du contrôle MonLibellé : si sa propriété Caption est "Joueur 1", elle est redéfinie à "Joueur 2", et inversement.



Ne confondez pas le libellé d'un contrôle (propriété Caption) et son nom (propriété Name). Le nom d'un contrôle sert à faire référence à ce contrôle dans le code du programme, et est invisible pour l'utilisateur final.

ControlTipText

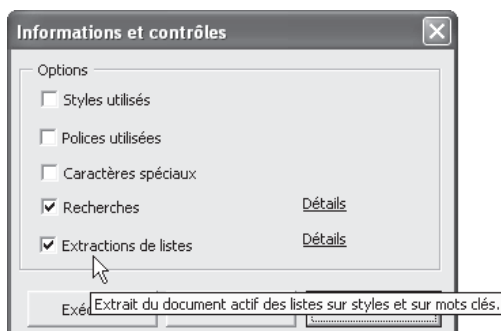


Cette propriété détermine le texte de l'info-bulle qui s'affiche lorsque l'utilisateur place le pointeur au-dessus du contrôle sans cliquer. La propriété ControlTipText accepte une valeur de type String (chaîne de caractères), et sa valeur par défaut est une chaîne vide. N'hésitez pas à utiliser cette propriété pour affecter des info-bulles décrivant brièvement la fonction des contrôles d'une feuille.

À la Figure 13.6, le pointeur a été placé au-dessus d'un contrôle dont la propriété ControlTipText a été définie afin de renseigner l'utilisateur sur sa fonction.

Figure 13.6

Les info-bulles permettent de renseigner l'utilisateur sur la fonction d'un contrôle.



ForeColor

Cette propriété détermine la couleur de premier plan du contrôle. La couleur de premier plan d'un contrôle correspond à la couleur d'affichage de sa propriété Caption.

SpecialEffect

Cette propriété détermine l'apparence du contrôle sur la feuille. La valeur acceptée par la propriété `SpecialEffect` varie selon le contrôle :

- Les contrôles `CheckBox` (case à cocher), `RadioButton` (bouton radio) et `ToggleButton` (bouton à bascule) acceptent l'une des constantes suivantes :
 - `fmButtonEffectFlat` ;
 - `fmButtonEffectSunken`.
- Les autres contrôles acceptent l'une des constantes suivantes :
 - `fmSpecialEffectFlat` ;
 - `fmSpecialEffectRaised` ;
 - `fmSpecialEffectSunken` ;
 - `fmSpecialEffectEtched`.

La valeur attribuée par défaut à la propriété `SpecialEffect` d'un contrôle correspond à l'apparence habituelle pour ce type d'objets dans les applications Windows. Il est donc préférable, pour ne pas dérouter l'utilisateur final, de conserver les valeurs attribuées par défaut. Pour plus d'informations sur les effets disponibles pour les contrôles, consultez l'aide Visual Basic (sélectionnez le nom de la propriété dans la fenêtre Propriétés et tapez sur F1).

Style



Cette propriété détermine la façon dont la sélection s'effectue dans un contrôle `Combo-Box`. L'utilisateur peut ou non être autorisé à saisir une valeur ne figurant pas dans la liste affectée au contrôle. La propriété `Style` accepte pour valeur l'une des deux constantes `fmStyle` :

- **`fmStyleDropDownCombo`** (par défaut). L'utilisateur peut sélectionner une valeur dans la zone de liste déroulante ou saisir manuellement une valeur de son choix. C'est, par exemple, le cas de la zone de liste `Zoom` de la barre d'outils Standard d'Excel.
- **`fmStyleDropDownList`**. L'utilisateur ne peut saisir de valeur dans la zone d'édition du contrôle. Les seules valeurs autorisées sont celles figurant dans la liste attachée au contrôle. C'est le cas des zones de listes déroulantes d'alignement de la boîte de dialogue `Cellules` d'Excel. Lorsque l'utilisateur saisit un caractère au clavier, le premier

élément de la liste commençant par ce caractère est sélectionné. Si aucun élément de la liste ne commence par le caractère saisi, la valeur du contrôle reste inchangée.

Utilisez un contrôle `ComboBox` dont la propriété `Style` est définie à `fmStyleDropDownList`, plutôt qu'un contrôle `ListBox`, pour réduire l'espace occupé par le contrôle sur la feuille.



Lorsque vous autorisez la saisie manuelle de valeurs dans la zone d'édition d'un contrôle `ComboBox`, assurez-vous que l'utilisateur ne pourra y saisir une valeur non valide, qui générerait une erreur ou aboutirait à un mauvais fonctionnement du programme. Affectez pour cela une procédure à l'événement `Change` du contrôle, qui s'assurera de la validité de la valeur saisie (voir l'exemple donné pour le contrôle `TextBox`, au chapitre suivant).

Value



Cette propriété détermine la valeur d'un contrôle. La propriété `Value` accepte des valeurs qui varient d'un contrôle à l'autre. La valeur d'un contrôle peut en effet correspondre à un état (le fait qu'une case soit ou non cochée) ou à un contenu (le texte saisi dans la zone d'édition d'une zone de texte), et peut donc accepter une valeur numérique ou de type `String`.

Le tableau suivant présente les valeurs acceptées par la propriété `Value` des contrôles standard :

Contrôle	Valeur acceptée par la propriété Value
CheckBox, OptionButton ou ToggleButton	Valeur de type <code>Boolean</code>, indiquant l'état du contrôle : <code>True</code> s'il est activé et <code>False</code> s'il est désactivé. Les contrôles <code>CheckBox</code> (case à cocher) et <code>OptionButton</code> (bouton d'option) sont activés lorsqu'ils sont cochés ; un contrôle <code>ToggleButton</code> (bouton bascule) est activé lorsqu'il est en surbrillance et enfoncé. Si leur propriété <code>TriState</code> est définie à <code>True</code>, ces contrôles acceptent un troisième état : ni activé ni inactivé. Leur propriété <code>Value</code> prend alors la valeur <code>Null</code>. Une case à cocher ou un bouton d'option dont la propriété <code>Value</code> est nulle apparaît estompé et contient une marque d'activation. Lorsqu'un bouton bascule est à l'état <code>Null</code>, son libellé apparaît estompé. <i>Voir aussi</i> la propriété <code>TriState</code>.
TextBox	Valeur de type <code>String</code> représentant le texte apparaissant dans la zone d'édition du contrôle.

<i>Contrôle</i>	<i>Valeur acceptée par la propriété Value</i>
ComboBox et ListBox	Valeur représentant l'élément sélectionné dans la liste des éléments du contrôle. La propriété <code>BoundColumn</code> du contrôle détermine son type de valeur. Il peut s'agir de la <i>valeur d'index</i> de l'élément sélectionné (sa position dans la liste) ou d'une chaîne correspondant au texte sélectionné. Dans le cas d'un contrôle à colonne multiple, il peut s'agir du texte de n'importe laquelle des colonnes. La propriété <code>Value</code> ne peut être utilisée pour une zone de liste à sélection multiple (propriété <code>MultiSelect</code>). Pour plus de précisions, reportez-vous aux sections traitant de ces contrôles, plus loin dans ce chapitre. <i>Voir aussi</i> la propriété <code>BoundColumn</code>.
ScrollBar et SpinButton	Valeur de type <code>Integer</code> comprise entre les valeurs des propriétés <code>Min</code> et <code>Max</code> du contrôle. La propriété <code>Value</code> d'un contrôle <code>ScrollBar</code> indique la position du curseur sur la barre de défilement. Elle vaut <code>Min</code> lorsque le curseur est tout en haut (barre de défilement horizontale) ou tout à gauche (barre de défilement verticale) de la barre, et <code>Max</code> lorsque le curseur est tout en bas ou à droite du contrôle. La propriété <code>Value</code> d'un contrôle <code>SpinButton</code> représente la valeur sélectionnée par l'utilisateur par un clic sur les boutons du contrôle. Lorsque l'utilisateur clique sur la flèche de défilement Haut ou Gauche du contrôle, sa propriété <code>Value</code> se rapproche de la valeur de <code>Min</code>. Lorsqu'il clique sur la flèche de défilement Bas ou Droite, la propriété <code>Value</code> se rapproche de la valeur de <code>Max</code>. <i>Voir aussi</i> les propriétés <code>Min</code> et <code>Max</code>.
MultiPage	Valeur de type <code>Integer</code> représentant la page sélectionnée. La première page d'un contrôle <code>MultiPage</code> correspond à la valeur 0. La valeur de la propriété <code>Value</code> de ce type de contrôle doit donc être comprise entre 0 et le nombre de pages moins 1.

La procédure suivante se déclenche lorsque l'utilisateur clique sur le bouton de commande `btnOK`. Elle teste la valeur de la propriété `Value` du contrôle `txtMotDePasse`, une zone de texte dans laquelle l'utilisateur est invité à saisir son mot de passe. Si aucun mot de passe n'a été saisi (la propriété `Value` du contrôle est égale à une chaîne vide), la boîte de dialogue de la Figure 13.7 s'affiche.

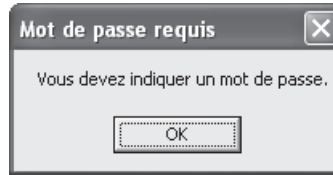
```

If txtMotDePasse.Value = "" Then
    MsgBox "Vous devez indiquer un mot de passe.", vbOKOnly, "Mot de passe requis"
Else
    Instructions
End If

```

Figure 13.7

Value est l'une des propriétés les plus utilisées.



Visible



Cette propriété détermine la visibilité d'un contrôle pour l'utilisateur. Un contrôle peut ainsi être rendu invisible à l'utilisateur à tout moment de l'exécution du programme. La propriété `Visible` accepte une valeur de type Boolean :

- **True** (par défaut). Indique que le contrôle est visible.
- **False**. Indique que le contrôle est masqué et n'apparaît pas sur la feuille.

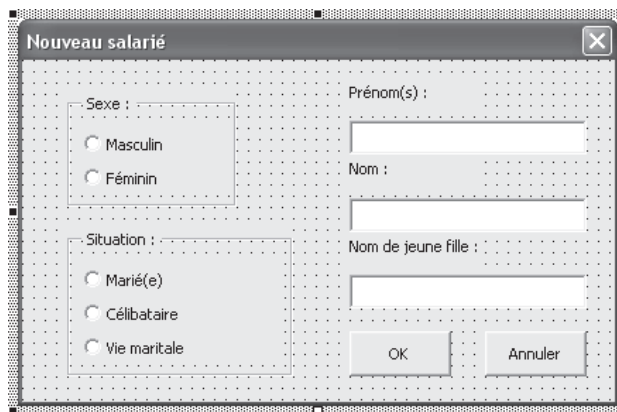


La propriété `Value` d'un contrôle n'influe pas sur son apparence durant la phase de conception. Le contrôle est toujours visible dans Visual Basic Editor.

La propriété `Visible` permet d'élaborer des feuilles dont l'apparence et les fonctionnalités varieront selon le contexte. Vous pouvez ainsi n'afficher les contrôles d'une feuille que s'ils doivent être renseignés par l'utilisateur. La Figure 13.8 présente la feuille `Nouveau salarié`, en phase de conception dans Visual Basic Editor.

Figure 13.8

La feuille `Nouveau salarié` en phase de conception.



Outre le sexe et la situation, l'utilisateur est invité à entrer un prénom et un nom pour le nouveau salarié. Une zone de texte et un libellé Nom de jeune fille permettent aussi d'indiquer un éventuel nom de jeune fille.

Les propriétés des contrôles `OptionButton` (bouton d'option), `CheckBox` (case à cocher) et `Label` (intitulé) de la feuille ont été définies ainsi :

<i>Propriété Name</i>	<i>Propriété Caption</i>
Contrôles <code>OptionButton</code>	
<code>optMasculin</code>	Masculin
<code>OptFéminin</code>	Féminin
<code>optMarié</code>	Marié(e)
<code>optCélibataire</code>	Célibataire
<code>optMaritale</code>	Vie maritale
Contrôles <code>Label</code>	
<code>lbPrénom</code>	Prénom(s) :
<code>lbNom</code>	Nom :
<code>lbNomJF</code>	Nom de jeune fille
Contrôles <code>TextBox</code>¹	
<code>txtPrénom</code>	
<code>txtNom</code>	
<code>txtNomJF</code>	

1. Les contrôles `TextBox` n'ont pas de propriété `Caption`.

Il est évidemment inutile d'afficher le libellé Nom de jeune fille et la zone de texte correspondante si les boutons d'option Féminin et Marié(e) ne sont pas tous deux cochés. La propriété `Visible` de ces contrôles a donc été définie à `False` afin qu'ils soient masqués à l'affichage de la feuille – la propriété `Value` du contrôle `optMasculin` étant définie à `True` afin que ce bouton soit coché par défaut. Le code suivant affiche les contrôles `txtNomJF` et `lbJF` lorsque cela est nécessaire et les masque, dans le cas contraire.

```
Private Sub optFéminin_Click()
    If optMarié.Value = True Then
        txtNomJF.Visible = True
        lbJF.Visible = True
    Else
```

```
        txtNomJF.Visible = False
        lbJF.Visible = False
    End If
End Sub

Private Sub optMarié_Click()
    If optFéminin.Value = True Then
        txtNomJF.Visible = True
        lbJF.Visible = True
    Else
        txtNomJF.Visible = False
        lbJF.Visible = False
    End If
End Sub

Private Sub optMasculin_Click()
    txtNomJF.Visible = False
    lbJF.Visible = False
End Sub

Private Sub optCélibataire_Click()
    txtNomJF.Visible = False
    lbJF.Visible = False
End Sub

Private Sub optMaritale_Click()
    txtNomJF.Visible = False
    lbJF.Visible = False
End Sub
```

La procédure `optFéminin_Click` se déclenche lorsque l'utilisateur clique sur le contrôle nommé `optFéminin` (le bouton d'option libellé `Féminin`). Si la propriété `Value` du contrôle `optMarié` est définie à `True` – le bouton `Marié` étant activé –, la propriété `Visible` des contrôles `txtNomJF` et `lbJF` est définie à `True`, et les contrôles apparaissent sur la feuille.

La procédure `optMarié_Click` se déclenche lorsque l'utilisateur clique sur le contrôle nommé `optMarié`, et opère de façon semblable en interrogeant la propriété `Value` du contrôle `optFéminin`. Si la valeur renvoyée est `True`, alors la propriété `Visible` des contrôles `txtNomJF` et `lbJF` est définie à `True`.

Les trois procédures suivantes se déclenchent lorsque l'utilisateur clique sur les contrôles `optMasculin`, `optCélibataire` et `optMaritale`. La propriété `Visible` des contrôles `txtNomJF` et `lbJF` est alors définie à `False`, et les contrôles apparaissent sur la feuille.

Figure 13.9

La propriété Visible permet de déterminer les conditions de visibilité des contrôles pour l'utilisateur.

Comportement

AutoSize

Cette propriété spécifie si un contrôle se redimensionne automatiquement pour afficher son contenu. Le contenu d'un contrôle désigne la valeur de sa propriété Caption ou, dans le cas d'un contrôle TextBox ou ComboBox, le texte qui apparaît dans sa zone d'édition. La propriété AutoSize accepte une valeur de type Boolean :

- **True.** Le contrôle se redimensionne automatiquement.
- **False** (par défaut). La taille du contrôle est fixe.

Info

Lorsque vous définissez la propriété AutoSize d'un contrôle durant la phase de conception, il se redimensionne automatiquement, en fonction de son contenu. Vous pouvez si vous le souhaitez définir une taille supérieure pour le contrôle. En phase d'exécution, le contrôle conservera la taille ainsi définie, tant que celle-ci suffira à en afficher tout le contenu.

La Figure 13.10 présente deux contrôles TextBox dont les propriétés AutoSize sont respectivement définies sur True et False. L'un s'est redimensionné automatiquement pour faire apparaître le texte saisi par l'utilisateur, tandis que le texte saisi dans le contrôle dont la taille est fixe est rogné.

Figure 13.10

La propriété AutoSize permet d'éviter que le texte ne soit rogné.

 Conseil

Ne définissez la propriété `AutoSize` d'un contrôle `TextBox` sur `True`, que si le nombre de caractères autorisés dans ce contrôle (propriété `MaxLenght`) est limité. Dans le cas contraire, l'utilisateur peut saisir un nombre illimité de caractères, et le contrôle peut être dimensionné jusqu'à atteindre les limites de la feuille.

AutoTab



Disponible pour les contrôles `TextBox` et `ComboBox` dont la propriété `MaxLenght` a été définie, la propriété `AutoTab` spécifie si une tabulation automatique se déclenche lorsque le nombre de caractères maximal autorisé dans le contrôle est atteint. Si tel est le cas, le *focus* est passé au contrôle suivant dans l'ordre de tabulation.

 Info

Le contrôle qui a le focus est le contrôle réceptif aux événements utilisateur (souris ou clavier). Un seul contrôle peut avoir le focus à un moment donné, et le type d'événements utilisateur reconnus varie d'un contrôle à l'autre.

Par exemple, taper sur la touche Entrée alors qu'un bouton de commande a le focus revient à cliquer sur ce bouton. La frappe de la touche Entrée n'aura aucun effet si le focus correspond à une case à cocher, alors que chaque frappe de la touche Tab le fera passer d'un état à l'autre.

 Définition

L'ordre de tabulation désigne l'ordre dans lequel le focus est transmis aux différents contrôles d'une feuille et est déterminé par la propriété `TabIndex` des contrôles.

La propriété `AutoTab` accepte une valeur de type `Boolean` :

- **True.** Le focus est passé au contrôle suivant dans l'ordre de tabulation lorsque le nombre de caractères maximal est atteint.
- **False** (par défaut). Lorsque le nombre de caractères maximal est atteint, les événements clavier demeurent sans effet, mais le contrôle `TextBox` garde le focus.

La propriété `AutoTab` sera utilisée judicieusement avec une zone de texte devant toujours recevoir un nombre de caractères fixe, comme un mot de passe ou un code produit. Le contrôle suivant sera ainsi sélectionné automatiquement sans que l'utilisateur ait à quitter le clavier.

AutoWordSelect

Cette propriété détermine la façon dont le texte est sélectionné dans la zone d'édition d'un contrôle TextBox, ou ComboBox. L'unité de sélection peut être le mot ou le caractère. La propriété AutoWordSelect accepte une valeur de type Boolean :

- **True** (par défaut). Lorsque l'utilisateur étend la sélection au-delà d'un mot, le mot entier est sélectionné et la sélection se fait ensuite mot par mot.
- **False**. La sélection se fait caractère par caractère, quelle que soit son étendue.

Cancel



Cette propriété spécifie si un contrôle CommandButton est ou non le bouton Annuler de la feuille. Lorsqu'un bouton de commande est le bouton Annuler de la feuille, la frappe de la touche Échap déclenche l'événement Click du bouton, qu'il ait ou non le focus. Outre cet accès, le bouton peut toujours être activé par les méthodes classiques – touche de raccourci, clic de souris ou frappe de la touche Entrée lorsque le contrôle a le focus. La propriété Cancel accepte une valeur de type Boolean :

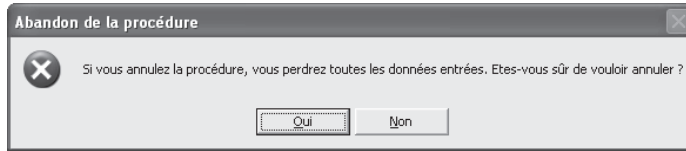
- **True**. Le bouton de commande est le bouton Annuler de la feuille.
- **False** (par défaut). Le bouton de commande n'est pas le bouton Annuler.

Dans le cas d'une procédure complexe, ou nécessitant un grand nombre d'opérations de la part de l'utilisateur, protégez l'activation du bouton d'annulation en affichant une boîte de dialogue demandant la confirmation de l'abandon. La procédure suivante affiche la boîte de dialogue présentée à la Figure 13.11 lorsque l'utilisateur active le bouton de commande cmdAnnuler.

```
Private Sub cmdAnnuler_Click()
    Dim Confirm As Single
    Confirm = MsgBox("Si vous annulez la procédure, vous perdrez toutes les données entrées. " _
        & "Etes-vous sûr de vouloir annuler ?", vbYesNo + vbCritical, "Abandon de la procédure")
    If Confirm = vbYes Then
        Exit Sub
    End If
End Sub
```

Figure 13.11

Protégez l'utilisateur d'un abandon involontaire de la procédure par l'affichage d'un message de confirmation.



Sur une même feuille, un seul bouton de commande peut être le bouton Annuler. Lorsque vous définissez la propriété `Cancel` d'un bouton de commande sur `True`, cette propriété est automatiquement définie à `False` pour tous les autres boutons de commande de la feuille.

Default



Cette propriété spécifie si un contrôle `CommandButton` est ou non le bouton par défaut de la feuille. Lorsqu'un bouton de commande est le bouton par défaut, la frappe de la touche Entrée déclenche l'événement `Click` du bouton, à condition qu'aucun autre bouton de commande de la feuille n'ait le focus. Outre cet accès, le bouton peut toujours être activé par les méthodes classiques – touche de raccourci, clic de souris ou frappe de la touche Entrée lorsque le contrôle a le focus. La propriété `Default` accepte une valeur de type `Boolean` :

- **True.** Le bouton de commande est le bouton par défaut.
- **False** (par défaut). Le bouton de commande n'est pas le bouton par défaut.



Sur une même feuille, un seul bouton de commande peut être le bouton par défaut. Lorsque vous définissez la propriété `Default` d'un bouton de commande sur `True`, cette propriété est automatiquement définie à `False` pour tous les autres boutons de commande de la feuille.

Enabled



Cette propriété détermine l'accessibilité d'un contrôle. Un contrôle est dit *accessible* lorsqu'il peut avoir le focus, et *inaccessible*, dans le cas contraire. Lorsqu'un contrôle est inaccessible, l'utilisateur ne peut le sélectionner ni à l'aide de la souris ni à l'aide de la touche Tab et il apparaît estompé. La propriété `Enabled` accepte une valeur de type `Boolean` :

- **True** (par défaut). Le contrôle est accessible.
- **False.** Le contrôle n'est pas accessible.

Utilisez la propriété `Enabled` pour déterminer l'accessibilité d'un contrôle en fonction du contexte. Dans l'exemple suivant, nous avons ajouté un contrôle `CheckBox` (case à cocher) sur la feuille `Nouveau salarié`, afin d'indiquer si le nouveau salarié est dégagé ou non des obligations militaires – sa propriété `Name` a été définie à `chkMilitaire`. Il est évidemment inutile que l'utilisateur puisse accéder à ce contrôle si le nouveau salarié est une femme. Nous avons ajouté deux instructions (en gras, dans le listing) afin de désactiver l'accès au contrôle lorsque le nouveau salarié est une femme et de l'activer s'il s'agit d'un homme.

```
Private Sub optFéminin_Click()  
    If optMarié.Value = True Then  
        txtNomJF.Visible = True  
        lbJF.Visible = True  
    Else  
        txtNomJF.Visible = False  
        lbJF.Visible = False  
    End If  
    chkMilitaire.Enabled = False  
End Sub  
  
Private Sub optMarié_Click()  
    If optFéminin.Value = True Then  
        txtNomJF.Visible = True  
        lbJF.Visible = True  
    Else  
        txtNomJF.Visible = False  
        lbJF.Visible = False  
    End If  
End Sub  
  
Private Sub optMasculin_Click()  
    txtNomJF.Visible = False  
    lbJF.Visible = False  
    chkMilitaire.Enabled = True  
End Sub  
  
Private Sub optCélibataire_Click()  
    txtNomJF.Visible = False  
    lbJF.Visible = False  
End Sub  
  
Private Sub optMaritale_Click()  
    txtNomJF.Visible = False  
    lbJF.Visible = False  
End Sub
```



Notez qu'un contrôle peut être accessible et non modifiable. Pour plus de précisions, voyez la propriété `Locked`.

Figure 13.12

La propriété `Enabled` permet de déterminer l'accessibilité d'un contrôle en fonction du contexte.

EnterKeyBehavior

Cette propriété détermine le comportement d'un contrôle `CheckBox` lorsque l'utilisateur appuie sur la touche Entrée. La propriété `EnterKeyBehavior` accepte une valeur de type `Boolean` :

- **True.** Une pression sur la touche Entrée crée une nouvelle ligne.
- **False** (par défaut). Une pression sur la touche Entrée passe le focus au contrôle suivant dans l'ordre de tabulation.



Si la propriété `MultiLine` d'un contrôle `CheckBox` est définie sur `False`, le contrôle n'autorise qu'une ligne, et une pression sur la touche Entrée entraînera toujours le passage du focus au contrôle suivant dans l'ordre de tabulation.

HideSelection

Cette propriété détermine si la sélection dans un contrôle `TextBox` ou `ComboBox` demeure ou non visible lorsque le contrôle n'a pas le focus. La propriété `HideSelection` accepte une valeur de type `Boolean` :

- **True** (par défaut). La sélection n'est visible que lorsque le contrôle a le focus.
- **False.** La sélection demeure visible, que le contrôle ait le focus ou non.

Maintenir la sélection visible dans un contrôle TextBox ou ComboBox lorsque celui-ci n'a pas le focus est déroutant pour l'utilisateur, car cette propriété est généralement spécifique du contrôle ayant le focus.

Locked



Cette propriété détermine si un contrôle peut ou non être modifié ou activé par l'utilisateur. La propriété Locked accepte une valeur de type Boolean :

- **True**. Le contrôle peut être modifié.
- **False** (par défaut). Le contrôle ne peut pas être modifié.



Utilisez la propriété Locked conjointement avec la propriété Enabled afin, par exemple, de permettre à l'utilisateur de copier le texte d'une zone de texte sans l'autoriser à en modifier le contenu.

La propriété Locked doit être définie sur True pour un contrôle ayant une fonction de consultation ; par exemple, un contrôle TextBox affichant le texte sélectionné dans le document ou le résultat d'un calcul.



Lorsque la propriété Locked d'un contrôle est définie à True, le contenu de ce contrôle ne peut être modifié directement sur la feuille durant la phase Conception. Si vous souhaitez modifier sa propriété Caption ou Value, vous devez le faire dans la fenêtre Propriétés.

MaxLength



Cette propriété spécifie le nombre maximal de caractères que l'utilisateur est autorisé à saisir dans un contrôle TextBox ou ComboBox. La propriété MaxLength accepte une valeur de type Integer, correspondant au nombre de caractères autorisés. Par défaut, la propriété MaxLength est définie sur 0, l'utilisateur étant alors autorisé à entrer un nombre de caractères indéterminé.

Lorsque le nombre de caractères maximal est atteint, les frappes de clavier restent sans effet. Vous pouvez aussi choisir que le focus soit passé au contrôle suivant dans l'ordre de tabulation en définissant la propriété AutoTab sur True.

**Conseil**

Affectez une valeur `MaxLength` aux contrôles `TextBox` destinés à recevoir un nombre de caractères constant (saisie d'un mot de passe ou d'un code produit, par exemple), ou lorsque la limitation du nombre de caractères est une nécessité du programme.

Voir aussi `AutoTab`.

MultiLine



Cette propriété détermine si le texte saisi dans un contrôle `TextBox` s'affiche sur la ligne suivante lorsqu'il arrive en bout de ligne. Si tel est le cas, le texte s'affiche sur la ligne suivante tant que les dimensions du contrôle le permettent. La propriété `MultiLine` accepte une valeur de type `Boolean` :

- **True** (par défaut). Le texte s'affiche sur plusieurs lignes lorsque cela est nécessaire.
- **False**. Le texte s'affiche sur une seule ligne.

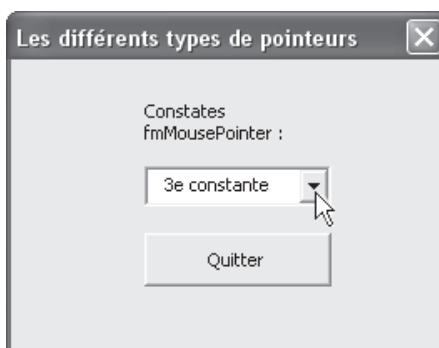
SelectionMargin

Cette propriété détermine si l'utilisateur peut ou non sélectionner une ligne de texte dans un contrôle `TextBox` ou `ComboBox` en plaçant le curseur à gauche de cette ligne. Si tel est le cas, le curseur se transforme en flèche orientée vers le haut et à droite lorsqu'il est placé à gauche de la ligne à sélectionner (voir Figure 13.13). La propriété `SelectionMargin` accepte une valeur de type `Boolean` :

- **True** (par défaut). Ce type de sélection est possible.
- **False**. Ce type de sélection est impossible.

Figure 13.13

Le curseur prend la forme d'une flèche indiquant qu'un clic entraînera la sélection de la ligne entière.



Style

Cette propriété détermine si un contrôle ComboBox autorise ou non la saisie d'une valeur ne figurant pas dans les options de la liste déroulante qui lui est attachée. La propriété Style accepte pour valeur une constante fmStyle :

- **fmStyleDropDownCombo** (par défaut). Le contrôle se comporte comme une zone de texte. L'utilisateur peut saisir une valeur de son choix dans la zone d'édition ou sélectionner l'une des options de la liste déroulante. Si les premières lettres saisies sont rapprochées de l'une des options de la liste, le complément automatique est proposé. L'utilisateur peut alors accepter le complément proposé ou saisir une valeur différente.
- **fmStyleDropDownList**. Le contrôle se comporte comme une zone de liste. L'utilisateur ne peut sélectionner qu'une des options de la liste. Il est impossible de saisir une valeur différente dans la zone d'édition du contrôle. Si l'utilisateur saisit une lettre au clavier alors que le contrôle a le focus, l'option de la liste la plus proche de la lettre saisie est sélectionnée.



La façon dont le rapprochement est effectué entre la valeur saisie par l'utilisateur dans la zone d'édition et les options de la liste est déterminée par les propriétés MatchEntry et MatchRequired du contrôle.

TabKeyBehavior

Cette propriété détermine si l'utilisateur peut ou non saisir une tabulation dans la zone d'édition d'un contrôle TextBox. La propriété TabKeyBehavior accepte une valeur de type Boolean :

- **True**. L'utilisateur peut saisir une tabulation dans la zone d'édition. La touche Tab ne peut donc pas être utilisée pour passer le focus au contrôle suivant dans l'ordre de tabulation.
- **False** (par défaut). La touche Tab entraîne le passage du focus au contrôle suivant dans l'ordre de tabulation.



Ne définissez cette propriété sur True que si l'exploitation des données du contrôle par le programme prévoit la gestion des tabulations.

TextAlign

Cette propriété détermine la façon dont le libellé d'un contrôle Label, ou le texte de la zone d'édition des contrôles TextBox et ComboBox, est aligné. La propriété TextAlign accepte l'une des trois constantes fmTextAlign :

- **fmTextAlignLeft** (par défaut). Aligne à gauche.
- **fmTextAlignCenter**. Aligne au centre.
- **fmTextAlignRight**. Aligne à droite.



Dans le cas d'un contrôle ComboBox, la propriété TextAlign ne détermine que l'alignement du texte saisi dans la zone d'édition, et non les entrées de la liste qui sont toujours alignées à gauche.

TripleState

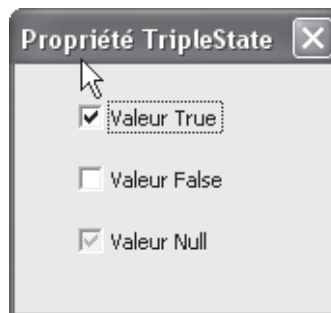


Cette propriété détermine si un contrôle OptionButton, CheckBox ou ToggleButton autorise ou non l'état null (ni activé ni désactivé). Une case à cocher ou un bouton d'option à l'état null apparaît estompé et contient une marque d'activation (voir Figure 13.14). Sa propriété Value renvoie alors la valeur Null. Lorsqu'un bouton bascule à l'état null, son libellé apparaît estompé. La propriété TripleState accepte une valeur de type Boolean :

- **True**. Le contrôle accepte l'état null.
- **False** (par défaut). Le contrôle n'accepte que deux états, activé ou désactivé.

Figure 13.14

Un contrôle CheckBox peut accepter trois états.



Voir aussi Value.

WordWrap

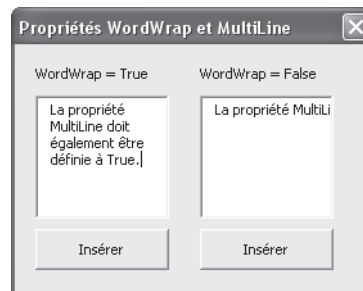
Cette propriété détermine si un contrôle autorise un changement de ligne afin d'afficher la totalité de son contenu. La propriété WordWrap est gérée par les contrôles possédant une propriété Caption et par le contrôle TextBox. Les contrôles ListBox et ComboBox ne gèrent pas cette propriété. La propriété WordWrap accepte une valeur de type Boolean :

- **True** (par défaut, sauf pour les contrôles CommandButton). Le contrôle autorise les changements de ligne.
- **False**. Le contrôle n'autorise pas les changements de ligne.

Lorsqu'un contrôle autorise un changement de ligne, celui-ci s'opère lorsque le contenu atteint la limite d'affichage de ligne et si la hauteur autorise la création d'une ligne supplémentaire. Si tel n'est pas le cas, aucun changement de ligne ne s'effectue et le texte se poursuit sur la même ligne. La Figure 13.15 présente deux contrôles TextBox dans lesquels un même texte a été saisi. La propriété WordWrap du premier est définie à True et celle du second, à False.

Figure 13.15

La propriété WordWrap permet de visualiser la totalité du contenu du contrôle, en autorisant l'affichage sur plusieurs lignes.



Si la propriété MultiLine d'un contrôle TextBox est définie sur False, une seule ligne est autorisée dans la zone d'édition, et la propriété WordWrap est sans effet.

Défilement

ScrollBars

Cette propriété détermine l'affichage d'une ou de plusieurs barres de défilement pour les feuilles, les contrôles TextBox et Frame. Ces contrôles peuvent posséder une barre de défilement horizontale, une barre de défilement verticale, ou les deux. Les barres de défilement

s'affichent lorsque les dimensions du contrôle ne permettent pas d'afficher la totalité de leur contenu. La propriété `ScrollBars` accepte l'une des constantes `fmScrollBars` :

- **`fmScrollBarsNone`**. Aucune barre de défilement ne s'affiche pour le contrôle.
- **`fmScrollBarsHorizontal`**. Le contrôle possède une barre de défilement horizontale.
- **`fmScrollBarsVertical`**. Le contrôle possède une barre de défilement verticale.
- **`fmScrollBarsBoth`**. Le contrôle possède des barres de défilement horizontale et verticale.



Les contrôles `ListBox` ne gèrent pas la propriété `ScrollBars`. Ils affichent toujours une barre de défilement verticale lorsque cela est nécessaire, afin que l'utilisateur puisse sélectionner les éléments de son choix dans la liste.

Un contrôle `TextBox` dont la propriété `MultiLine` est définie à `False` ne peut afficher de barre de défilement verticale, puisque son contenu est toujours affiché sur une seule ligne. Un contrôle `TextBox` dont la propriété `WordWrap` est définie à `True` ne peut afficher de barre de défilement horizontale, puisque, chaque fois que la limite d'affichage liée aux dimensions du contrôle est atteinte, le contenu s'affiche sur la ligne suivante.

Une barre de défilement placée sur une feuille ou sur un contrôle `MultiPage` permet l'affichage d'une zone supérieure à la taille réelle de la feuille ou du contrôle. Lorsque vous placez une barre de défilement sur un contrôle `MultiPage` ou sur une feuille, vous devez donc spécifier la taille totale de la zone pouvant être affichée à l'aide des barres de défilement (propriétés `ScrollHeight` et `ScrollWidth`). Elle doit évidemment être supérieure à la taille réelle du contrôle (propriété `Height`). La propriété `ScrollHeight` est, par conséquent, généralement définie dans une procédure, relativement à la valeur de la propriété `Height` (par exemple, `MonContrôle.ScrollHeight = MonContrôle.Height * 1.5`).

Pour plus de précisions sur l'utilisation des barres de défilement sur les feuilles et les contrôles `MultiPage`, reportez-vous à l'Aide Visual Basic.



Lorsque vous définissez à `True` les propriétés `MultiLine` et `WordWrap` d'un contrôle `TextBox`, affectez la valeur `fmScrollBarsVertical` à sa propriété `ScrollBars`. Une barre de défilement sera ainsi affichée lorsque les capacités d'affichage du contrôle seront dépassées.



Lorsque vous affectez une barre de défilement à un contrôle, assurez-vous que le contrôle dispose d'assez d'espace pour afficher ces barres.

KeepScrollsVisible

Cette propriété détermine si les barres de défilement d'un contrôle sont ou non visibles lorsqu'elles n'ont aucune utilité, c'est-à-dire lorsque la totalité du contenu du contrôle s'affiche. La propriété `ScrollBars` accepte l'une des constantes `fmScrollBars` :

- **fmScrollBarsNone.** Les barres de défilement ne sont affichées que lorsque les dimensions du contrôle ne permettent pas d'en visualiser le contenu.
- **fmScrollBarsHorizontal.** La barre de défilement horizontale est toujours affichée, quel que soit le contenu du contrôle.
- **fmScrollBarsVertical.** La barre de défilement verticale est toujours affichée, quel que soit le contenu du contrôle.
- **fmScrollBarsBoth.** Les barres de défilement horizontale et verticale sont toujours affichées, quel que soit le contenu du contrôle.

Delay

Cette propriété détermine la périodicité avec laquelle les événements `SpinUp`, `SpinDown` et `Change` sont reconnus sur un contrôle `SpinButton` ou `ScrollBar`, lorsque l'utilisateur clique sur les flèches de défilement ou dans la barre de défilement du contrôle et maintient le bouton enfoncé.

La propriété `Delay` accepte une valeur numérique de type `Long`, qui représente la périodicité de déclenchement de l'événement en millisecondes. Sa valeur par défaut est 50.

Pour vous déplacer à l'aide d'une barre de défilement ou modifier une valeur à l'aide d'un bouton toupie, vous pouvez cliquer une fois sur le contrôle pour vous déplacer d'une unité ou maintenir le bouton de la souris enfoncé. Dans ce cas, un premier événement se déclenche. Un court laps de temps s'écoule. Puis, l'événement s'exécute en boucle à intervalle régulier jusqu'à ce que vous relâchiez le bouton de la souris. La propriété `Delay` correspond à l'intervalle de temps séparant chaque événement exécuté en série. Le laps de temps entre le premier événement et la série d'événements qui s'exécute lorsque l'utilisateur maintient le bouton de la souris enfoncé permet à l'utilisateur de générer un événement par un simple clic de souris, sans qu'une série d'événements ne se déclenche. Ce laps de temps est égal à cinq fois la valeur de la propriété `Delay`.

Conseil

Ne modifiez pas la valeur par défaut de la propriété `Delay`. Elle correspond au comportement habituel des barres de défilement et des boutons toupies en environnement Windows.

Une valeur trop importante diminue l'efficacité du contrôle en augmentant le délai entre chaque événement déclenché en série. Une valeur trop petite risque de déclencher une série d'événements au moindre clic, alors que l'utilisateur ne souhaite déclencher qu'un simple événement.

Max et Min



Les propriétés Max et Min d'un contrôle ScrollBar ou SpinButton déterminent les valeurs maximale et minimale que peut prendre le contrôle et acceptent une valeur de type Integer.

Lorsque vous déplacez le curseur de défilement d'un contrôle ScrollBar ou lorsque vous cliquez sur l'une des flèches de défilement d'un contrôle SpinButton, la propriété Value se déplace entre les valeurs de Max et de Min. La propriété Value d'un contrôle ScrollBar vaut Min lorsque le curseur est tout en haut (barre de défilement horizontale) ou tout à gauche (barre de défilement verticale) de la barre, et Max lorsque le curseur est tout en bas ou à droite du contrôle.

La propriété Value d'un contrôle SpinButton représente la valeur qu'a sélectionnée l'utilisateur en cliquant sur les flèches de défilement du contrôle. Lorsque l'utilisateur clique sur la flèche de défilement Haut ou Gauche du contrôle, sa propriété Value se rapproche de la valeur de Min. Lorsqu'il clique sur la flèche de défilement Bas ou Droite, la propriété Value se rapproche de la valeur de Max.

L'unité d'incréméntation de la propriété Value du contrôle est déterminée par sa propriété SmallChange, ainsi que par sa propriété LargeChange dans le cas d'un contrôle ScrollBar. C'est donc la conjugaison de ces trois propriétés qui détermine le nombre de valeurs que peut prendre le contrôle (représenté par le nombre de positions possibles pour le curseur sur la barre de défilement, dans le cas d'un contrôle ScrollBar). Ce nombre est égal à $[(\text{Max} - \text{Min}) / \text{SmallChange}]$ plus 1.

Si les propriétés Min et Max du contrôle sont respectivement définies à 0 et à 100, et la propriété SmallChange, à 10, le contrôle peut prendre onze valeurs (0, 10, ..., 100). Chaque clic sur la flèche de défilement Bas ou Droit du contrôle incrémentera sa valeur de 10, jusqu'à atteindre la valeur de Max. Les clics resteront alors sans effet. De la même façon, chaque clic sur la flèche de défilement Haut ou Gauche du contrôle ôtera 10 à sa valeur, jusqu'à atteindre la valeur de Min. Les clics resteront alors sans effet.



Les valeurs par défaut des propriétés Min et Max d'un contrôle ScrollBar sont respectivement 0 et 32 767. La valeur de ses propriétés SmallChange et LargeChange étant définie par défaut à 1, le contrôle ainsi déterminé accepte 32 768 positions.

Les valeurs par défaut des propriétés Min et Max d'un contrôle SpinButton sont respectivement 0 et 100. La valeur de sa propriété SmallChange étant définie par défaut à 1, le contrôle ainsi défini accepte 101 positions.

SmallChange



Cette propriété spécifie la valeur d'incrément de la propriété Value d'un contrôle ScrollBar ou SpinButton lors des événements SpinDown ou SpinUp. La propriété SmallChange accepte une valeur de type Integer.

Chaque fois que l'utilisateur clique sur la flèche de défilement Bas (contrôle vertical) ou Droite (contrôle horizontal) du contrôle, sa valeur est incrémentée de la valeur de SmallChange, jusqu'à atteindre la valeur de la propriété Max. Les clics restent alors sans effet.

Chaque fois que l'utilisateur clique sur la flèche de défilement Haut ou Gauche du contrôle, sa valeur est décrémentée de la valeur de SmallChange, jusqu'à atteindre la valeur de la propriété Min. Les clics restent alors sans effet.

Conjuguée aux propriétés Min et Max du contrôle, la propriété SmallChange détermine le nombre de valeurs possible pour le contrôle, correspondant au nombre de positions que peut prendre le curseur sur la barre de défilement dans le cas d'un contrôle ScrollBar.

Pour un exemple d'exploitation de ces propriétés, reportez-vous à la section consacrée à l'exploitation des objets SpinButton décrite au chapitre suivant.

LargeChange



Cette propriété spécifie la valeur d'incrément de la propriété Value d'un contrôle ScrollBar lorsque l'utilisateur clique dans la barre de défilement, entre le curseur et l'une des flèches de défilement. La propriété LargeChange accepte une valeur de type Integer. Sa valeur par défaut est 1.

Chaque fois que l'utilisateur clique dans la barre de défilement, entre le curseur et la flèche de défilement Bas (contrôle vertical) ou Droite (contrôle horizontal) du contrôle, sa valeur est incrémentée de la valeur de LargeChange, jusqu'à atteindre la valeur de la propriété Max. Les clics restent alors sans effet.

Chaque fois que l'utilisateur clique dans la barre de défilement, entre le curseur et la flèche Haut ou Gauche du contrôle, sa valeur est décrémentée de la valeur de LargeChange, jusqu'à atteindre la valeur de la propriété Min. Les clics restent alors sans effet.

Dans l'environnement Windows, un clic dans une barre de défilement entraîne généralement un déplacement supérieur à un clic sur la flèche de défilement. Par exemple, un clic sur la flèche de défilement verticale d'une fenêtre Excel génère un défilement d'une ligne, tandis qu'un clic dans la barre de défilement génère un défilement d'un écran.

Pour un exemple d'utilisation de la propriété LargeChange, voyez la section consacrée à l'exploitation des objets ScrollBar, au chapitre suivant.

Divers

Accelerator



Cette propriété permet de définir un raccourci clavier pour un contrôle. Elle accepte une valeur de type String correspondant au raccourci clavier du contrôle. Pour attribuer un raccourci clavier à un contrôle, affectez un des caractères formant le texte de sa propriété Caption à sa propriété Accelerator.

La touche de raccourci du contrôle apparaît soulignée sur la feuille. Si la touche de raccourci apparaît plusieurs fois dans le libellé du contrôle, la première occurrence est soulignée. La casse de la lettre de raccourci affectée à un contrôle ne sera considérée que si le libellé du contrôle contient à la fois la lettre majuscule et la lettre minuscule.

Combinée à la touche Alt, la saisie de la touche de raccourci d'un contrôle déclenche le passage du focus à ce contrôle. Si l'événement par défaut du contrôle est l'événement Click (pour un bouton de commande ou une case à cocher, par exemple), alors celui-ci est exécuté.

La Figure 13.16 présente une feuille dont tous les contrôles ont été affectés à un raccourci clavier.

Figure 13.16

N'hésitez pas à affecter des touches de raccourci aux contrôles d'une feuille afin d'en optimiser l'utilisation.

Lorsqu'une touche de raccourci est affectée à un contrôle Label, elle entraîne le passage du focus au contrôle suivant dans l'ordre de tabulation. Cette fonction est particulièrement intéressante pour affecter un raccourci clavier permettant d'activer un contrôle TextBox. Il suffit en effet d'attribuer une touche de raccourci au contrôle Label le légendant et de s'assurer que ce dernier précède le contrôle TextBox dans l'ordre de tabulation.



Veillez à ne pas attribuer un même raccourci clavier à des contrôles différents sur une feuille. Lorsqu'une même touche de raccourci est affectée à plusieurs contrôles, la saisie de ce raccourci entraîne le passage du focus au premier des contrôles suivant le contrôle actif, dans l'ordre de tabulation.

GroupName



Cette propriété permet d'associer des contrôles `OptionButton`, afin que l'utilisateur ne puisse valider que l'un d'eux. La propriété `GroupName` accepte une valeur de type `String`. Pour associer des contrôles `OptionButton`, affectez la même chaîne de caractères à leurs propriétés `GroupName` respectives.

Notez qu'il existe deux moyens d'associer des boutons d'option sur une feuille : les placer sur un même cadre (contrôle `Frame`) ou affecter une même valeur à leurs propriétés `GroupName` respectives. Cette seconde possibilité permet de gagner de la place sur la feuille, en y plaçant un contrôle de moins, et offre plus de liberté puisque les boutons d'option peuvent être placés n'importe où sur la feuille. L'utilisation d'un cadre est cependant souvent *visuellement* plus efficace, surtout lorsque la feuille propose plusieurs groupes de boutons d'option. La distinction des différents groupes est ainsi rendue évidente par la délimitation qu'offre le cadre.



Pour définir à l'identique la propriété `GroupName` d'un ensemble de boutons d'options, sélectionnez d'abord les contrôles concernés sur la feuille. Ouvrez ensuite la fenêtre Propriétés (touche F4) et affectez la valeur de votre choix à la propriété `GroupName`. Tous les contrôles sélectionnés sont affectés par cette modification.

HelpContextID

Cette propriété sert à spécifier une rubrique d'aide pour le contrôle. Elle accepte une valeur de type `Long` correspondant à l'identificateur de contexte de l'une des rubriques d'aide du fichier d'aide. Cet ouvrage n'aborde pas ce sujet.

MouseIcon

Cette propriété permet de spécifier une *icône souris* personnalisée pour le contrôle. L'icône souris d'un contrôle est l'apparence que prend le pointeur lorsqu'il est placé au-dessus du contrôle. Pour que cette propriété ait un effet, la propriété `MousePointer` doit être définie à `fmMousePointerCustom`. Pour affecter une icône personnalisée à un contrôle à partir de la fenêtre Propriétés :

1. Cliquez sur le bouton Parcourir de la propriété MouseIcon, dans la fenêtre Propriétés. La fenêtre Charger une image s'affiche.
2. Sélectionnez un fichier de format valide qui sera utilisé comme icône souris pour le contrôle (l'extension de fichier la plus courante est .ico), puis cliquez sur Ouvrir.

Vous pouvez aussi écrire du code permettant d'affecter une icône personnalisée à un contrôle en cours d'exécution d'un programme. Faites pour cela appel à la fonction LoadPicture, selon la syntaxe suivante :

```
Contrôle.MouseIcon = LoadPicture(NomIcône)
```

où *Contrôle* est le contrôle et *NomIcône*, le chemin et le nom de l'icône personnalisée.

Figure 13.17

La propriété MouseIcon permet d'affecter des icônes personnalisées aux contrôles.



Pour affecter un curseur personnalisé à un contrôle, vous devez également affecter la valeur `fmMousePointerCustom` à sa propriété `MousePointer` – que le curseur soit affecté au contrôle lors de la phase de conception de la feuille ou lors de l'exécution du code à l'aide de la fonction `LoadPicture`.

MousePointer

Cette propriété permet de déterminer l'apparence du pointeur de souris du contrôle. La propriété `MousePointer` accepte pour valeur l'une des dix-sept constantes `fmMousePointer`. Les seize premières correspondent à un type de pointeur particulier, et la dix-septième, `fmMousePointerCustom`, permet d'affecter un pointeur personnalisé au contrôle (voir section précédente).

Pour visualiser les différents types de pointeurs de souris disponibles, placez sur une feuille un contrôle `ComboBox` et un contrôle `CommandButton` et définissez leurs propriétés comme indiqué dans le tableau suivant :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmTestPointeur
Contrôle ComboBox	
Name	cboConstantes
Locked	True
Contrôle CommandButton	
Name	cmdQuitter
Value	Quitter

Placez ensuite le code suivant dans la section Déclarations de la feuille :

```
Private Sub UserForm_Initialize()
    Dim ValConstante
    For ValConstante = 1 To 16
        cboConstantes.AddItem (ValConstante & "e constante")
    Next ValConstante
End Sub

Private Sub cboConstantes_Click()
    cmdQuitter.MousePointer = cboConstantes.ListIndex
End Sub

Private Sub cmdQuitter_Click()
    fmTestPointeur.Hide
End Sub
```

Sélectionnez ensuite la feuille dans Visual Basic Editor, puis cliquez sur le bouton Exécuter de la barre d'outils Standard. La feuille s'affiche. Sélectionnez une à une les différentes options de la zone de liste déroulante et placez le pointeur sur le bouton pour chaque sélection, afin de visualiser les différents types de pointeurs disponibles. Cliquez sur le bouton Quitter pour fermer le programme et retourner à Visual Basic Editor.



Lorsque vous modifiez la propriété MousePointer d'un objet UserForm (feuille), le pointeur de souris déterminé affecte toutes les feuilles, quel que soit le contrôle au-dessus duquel se trouve le pointeur. Pour le vérifier, remplacez l'instruction

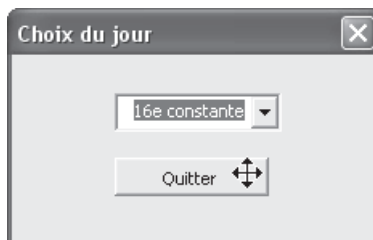
cmdQuitter.MousePointer = cboConstantes.ListIndex

par

fmTestPointeur.MousePointer = cboConstantes.ListIndex

Figure 13.18

Vous pouvez modifier l'apparence d'un pointeur à tout moment de l'exécution du programme.



TabIndex



Cette propriété détermine la position du contrôle dans l'ordre de tabulation de la feuille. Elle accepte une valeur de type Integer, comprise entre 0 et le nombre de contrôles de la feuille moins 1. (La valeur de la propriété TabIndex du premier contrôle dans l'ordre de tabulation est égale à 0.)



Pour un rappel des notions de focus et d'ordre de tabulation, reportez-vous à la section décrivant la propriété AutoTab, plus haut dans ce chapitre.

Lorsque l'utilisateur tape sur la touche Tab, le focus est passé au contrôle dont la valeur de la propriété TabIndex est égale à celle du contrôle actif plus 1. S'il utilise la combinaison Maj+Tab, le focus est passé au contrôle dont la valeur de la propriété TabIndex est égale à celle du contrôle actif moins 1.

Si la propriété Enabled du contrôle devant recevoir le focus est définie à False, il est ignoré, et le focus est passé au contrôle suivant, dans l'ordre de tabulation de la feuille. Si vous souhaitez qu'un contrôle soit ignoré dans l'ordre de tabulation tout en restant accessible à l'utilisateur, définissez sa propriété TabStop à False.

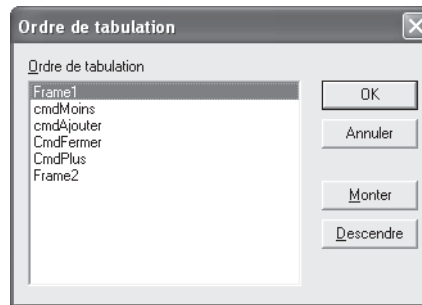
Pour définir l'ordre de tabulation d'une feuille en phase de conception, il est plus simple d'utiliser la boîte de dialogue Ordre de tabulation que de définir une à une les propriétés TabIndex des contrôles :

1. Cliquez du bouton droit sur la feuille (pas sur un contrôle) et, dans le menu contextuel qui s'affiche, choisissez la commande Ordre de tabulation.

La boîte de dialogue Ordre de tabulation s'affiche (voir Figure 13.19). Les contrôles de la feuille y sont répertoriés par leur nom (propriété Name) et classés selon leur position dans l'ordre de tabulation de la feuille (le contrôle dont la propriété TabIndex est égale à 0 se trouve tout en haut de la liste).

Figure 13.19

La boîte de dialogue Ordre de tabulation.



2. Pour déplacer un contrôle dans l'ordre de tabulation de la feuille, sélectionnez-le et cliquez sur le bouton Vers le haut, pour le faire remonter dans l'ordre de tabulation, ou sur le bouton Vers le bas, pour le faire descendre dans l'ordre de tabulation.

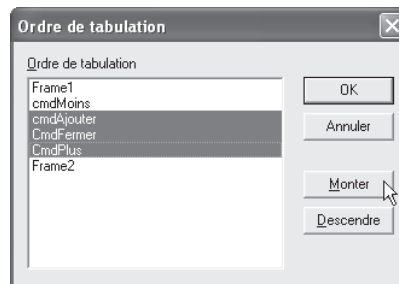
Chaque fois que vous déplacez un contrôle dans la boîte de dialogue Ordre de tabulation, sa propriété TabIndex et celle du contrôle dont il prend la place sont interverties.

3. Vous pouvez déplacer simultanément plusieurs contrôles dans la boîte de dialogue Ordre de tabulation, afin de modifier leur emplacement dans l'ordre de tabulation de la feuille, tout en conservant leurs positions relatives.

Pour sélectionner des contrôles contigus, cliquez sur le premier contrôle puis, tout en maintenant la touche Maj enfoncée, cliquez sur le dernier contrôle. Pour sélectionner des contrôles non contigus, cliquez sur les contrôles voulus tout en maintenant la touche Ctrl enfoncée.

Figure 13.20

Déplacez simultanément les contrôles dont vous souhaitez modifier la position dans l'ordre de tabulation, tout en leur conservant leurs positions relatives.



TabStop

Cette propriété détermine si un contrôle peut ou non recevoir le focus lorsque l'utilisateur tape sur la touche Tab ou sur la combinaison Maj+Tab. La propriété TabStop accepte une valeur de type Boolean :

- **True** (par défaut). Si sa position dans l'ordre de tabulation le justifie, le contrôle reçoit le focus lorsque la touche Tab ou la combinaison Maj+Tab est activée.
- **False**. Le focus ne peut être passé au contrôle par la touche Tab. Bien qu'il conserve sa propriété TabIndex, le contrôle est ignoré s'il est le suivant (touche Tab seule) ou le précédent (Maj+Tab) dans l'ordre de tabulation, et le focus est passé au contrôle suivant.

Tag

Cette propriété permet de stocker des informations supplémentaires sur le contrôle sous la forme d'une chaîne de caractères. Il peut s'agir d'une description du contrôle, qui pourra être affichée à l'intention de l'utilisateur si nécessaire.

Emplacement

Height et Width

Ces propriétés déterminent respectivement la hauteur et la largeur d'un contrôle. Les propriétés Height et Width acceptent une valeur de type Single représentant la hauteur et la largeur du contrôle en points.

Vous n'avez *a priori* pas à vous préoccuper de ces propriétés, puisqu'elles sont automatiquement mises à jour lorsque vous redimensionnez le contrôle dans Visual Basic Editor. Pour déterminer la taille de vos contrôles, préférez donc les méthodes de dimensionnement présentées au chapitre précédent.

Left et Top

Ces propriétés déterminent respectivement la position du contrôle par rapport au bord gauche et au bord supérieur de l'objet qui le contient. Les propriétés Left et Top acceptent une valeur de type Single représentant la distance séparant le contrôle des bords gauche et supérieur de l'objet conteneur en points.

Vous n'avez *a priori* pas à vous préoccuper de ces propriétés, puisqu'elles sont automatiquement mises à jour lorsque vous déplacez le contrôle dans Visual Basic Editor. Pour déterminer la position de vos contrôles sur la feuille, préférez donc les méthodes de mise en forme présentées au chapitre précédent.

Dans le cas d'une feuille, les propriétés Left et Top représentent la distance en points de la feuille par rapport aux bords gauche et supérieur de l'écran lors de son affichage. Ces propriétés ne sont considérées que si la propriété StartUpPosition de la feuille est définie à 0 (manuel).

StartUpPosition

Cette propriété détermine la position de la feuille sur l'écran lors de l'affichage de la feuille. La propriété StartUpPosition accepte l'une des valeurs suivantes :

- **0 - Manual1.** La position de la feuille sur l'écran est déterminée par ses propriétés Left et Top.
- **1 - CenterOwner** (par défaut). La feuille est centrée sur la fenêtre de l'application hôte.
- **2 - CenterScreen.** La feuille est centrée sur l'écran, quelles que soient la position et la taille de la fenêtre de l'application hôte.
- **3 - Windows Default.** La feuille s'affiche dans l'angle supérieur gauche de l'écran. Utilisez cette valeur lorsque vous souhaitez que la feuille affichée masque le moins possible le document actif.

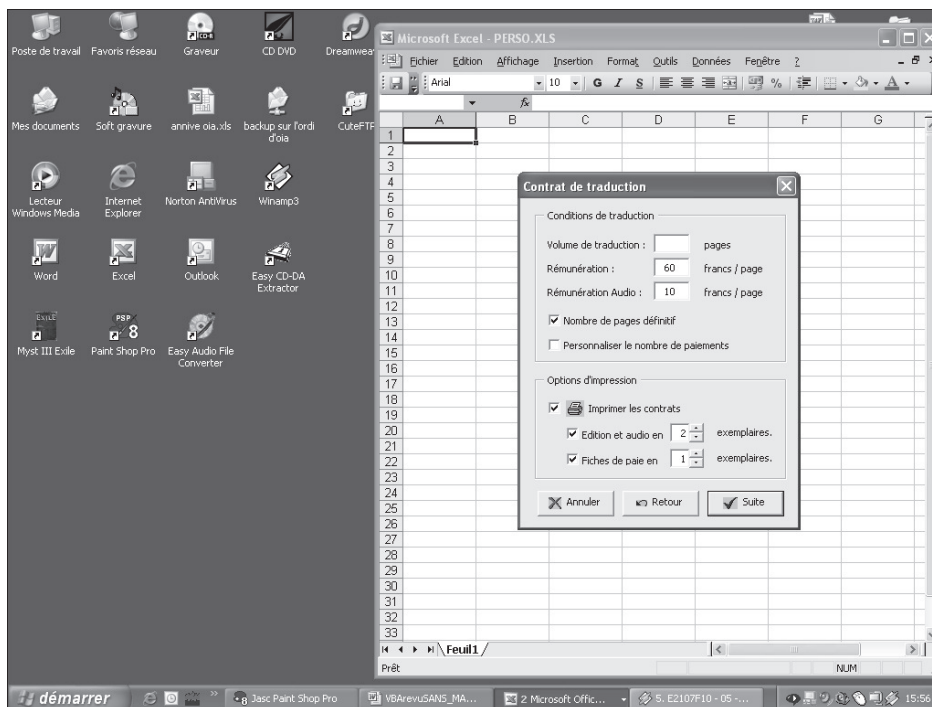
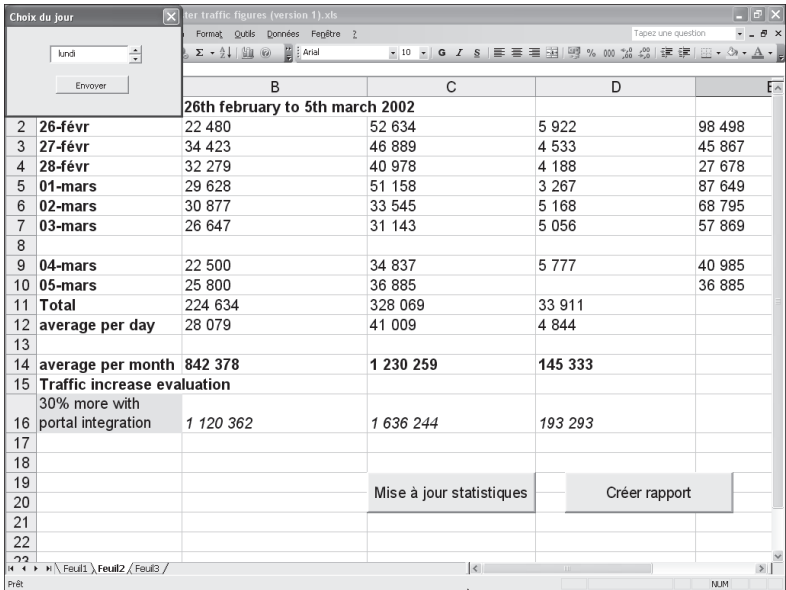


Figure 13.21

Vous pouvez choisir de toujours afficher la feuille au centre de la fenêtre de l'application hôte, quelles que soient la taille et la position de cette dernière.

Figure 13.22
L’affichage de la feuille dans l’angle supérieur gauche de l’écran permet d’éviter que des données importantes du document ne soient masquées.



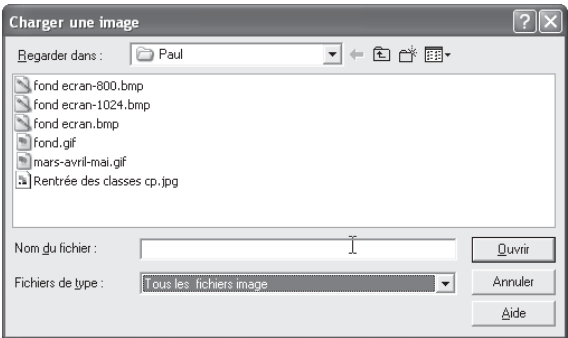
Image

Picture

Cette propriété permet d’affecter une image à un contrôle ou à une feuille. Pour affecter une image à un objet :

1. Affichez sa fenêtre Propriétés et cliquez sur le bouton Parcourir (...) de la propriété Picture. La boîte de dialogue Charger une image s’affiche (voir Figure 13.23).

Figure 13.23
Sélectionnez le fichier image que vous souhaitez affecter au contrôle.



2. Sélectionnez un fichier de format valide qui sera utilisé comme image pour le contrôle ou la feuille (.bmp et .ico sont des formats valides), puis cliquez sur Ouvrir.

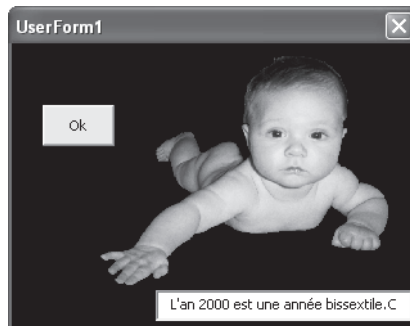
Vous pouvez aussi écrire du code permettant d'affecter une image à un objet en cours d'exécution d'un programme. Faites pour cela appel à la fonction LoadPicture, selon la syntaxe suivante :

```
Objet.Picture = LoadPicture(NomImage)
```

où Objet est la feuille ou le contrôle et NomImage, le chemin et le nom de l'image.

Figure 13.24

Vous pouvez donner un peu de fantaisie à vos feuilles en leur affectant des images.



Pour supprimer une image d'un contrôle, sélectionnez la valeur de la propriété Picture dans la fenêtre Propriétés et appuyez sur la touche Suppr.

PictureAlignment

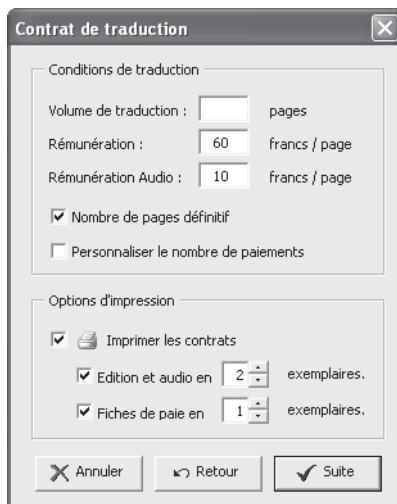
Cette propriété détermine l'emplacement d'une image (propriété Picture) sur un contrôle ou sur une feuille. La propriété PictureAlignment accepte pour valeur l'une des constantes fmPictureAlignment :

- **fmPictureAlignmentTopLeft**. L'image est placée dans l'angle supérieur gauche de l'objet.
- **fmPictureAlignmentTopRight**. L'image est placée dans l'angle supérieur droit de l'objet.
- **fmPictureAlignmentCenter** (par défaut). L'image est centrée sur l'objet.
- **fmPictureAlignmentBottomLeft**. L'image est placée dans l'angle inférieur gauche de l'objet.
- **fmPictureAlignmentBottomRight**. L'image est placée dans l'angle inférieur droit de l'objet.

Pour les contrôles `CommandButton`, la propriété `PictureAlignment` est remplacée par la propriété `PicturePosition`. Utilisez cette propriété pour affecter une image à un bouton de commande sans en masquer le libellé. Une image a été affectée à la propriété `Picture` des trois contrôles `CommandButton` de la Figure 13.25. Leur propriété `PicturePosition` a été définie sur `fmPicturePositionLeftCenter` afin de ne pas masquer leur libellé. Notez le contrôle (et non la propriété) `Picture` représentant une imprimante.

Figure 13.25

L'affectation d'une image à la propriété `Picture` des boutons de commande de vos feuilles leur donnera un peu de gaieté.



Astuce

Réalisez une capture d'écran d'une fenêtre dont vous souhaitez exploiter les icônes. Détourez ensuite ces dernières dans un logiciel de dessin tel que Paint Shop Pro et enregistrez-les au format BMP. Vous pourrez ensuite les exploiter dans vos feuilles VBA comme contrôle `Picture` ou comme propriété `Picture` d'un contrôle `CommandButton`.

PictureSizeMode

Cette propriété détermine de quelle façon l'image d'un contrôle ou d'une feuille s'affiche si sa taille diffère de celle de l'objet conteneur. Elle n'est pas gérée par le contrôle `CommandButton`. La propriété `PictureSizeMode` accepte pour valeur l'une des méthodes `PictureSizeMode` :

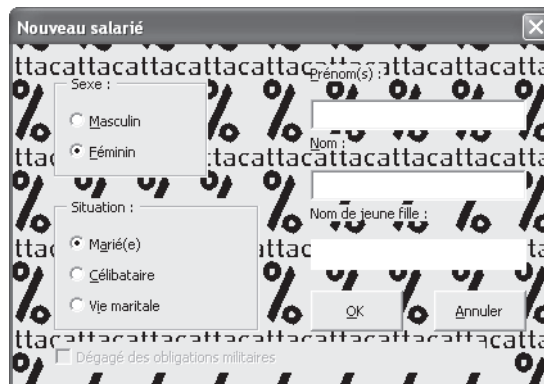
- **`fmPictureSizeModeClip`** (par défaut). Si la taille de l'image est supérieure à celle du contrôle, seule la partie tenant dans le contrôle s'affiche. La partie rognée variera en fonction de l'alignement de l'image, déterminé par la propriété `PictureAlignment`.

- ## PicturePosition

PictureTiling

- **True.** Si l'image est plus petite que l'objet conteneur, elle s'affiche en mosaïque sur la page (voir Figure 13.26). Pour que l'affichage en mosaïque s'effectue correctement, la propriété `PictureSizeMode` doit être définie à `fmPictureSizeModeClip`.
- **False** (par défaut). L'image n'est pas affichée en mosaïque.

*Une image affichée
en mosaïque sur une feuille.*





Une image affichée en mosaïque sur une feuille ou un contrôle risque d'être rognée sur le bord supérieur ou inférieur de l'objet, ainsi que sur son bord gauche ou droit. Il est en effet peu probable que la hauteur et la largeur de l'objet conteneur soient des multiples de la hauteur et de la largeur de l'image. Les bords de l'objet présentant des images rognées dépendront de l'alignement de l'image sur l'objet (propriété `PictureAlignment`).

Police

Font

Cette propriété détermine la police de caractères affectée à l'affichage du contenu d'un contrôle. La police par défaut est la police Tahoma Regular, corps 9. Pour modifier la propriété Font d'un contrôle, cliquez sur le bouton ... du volet droit de la fenêtre Propriétés. Dans la boîte de dialogue qui s'affiche, déterminez la police de caractères et les attributs voulus, puis cliquez sur OK.



14

Maîtriser le comportement des contrôles

Au sommaire de ce chapitre

- Créer des procédures événementielles
- Exemple d'exploitation des contrôles

Ce chapitre vous propose de découvrir comment tirer vraiment parti des interfaces utilisateur en leur ajoutant de l'interactivité grâce aux procédures événementielles. Il vous présente également, contrôle par contrôle, les techniques les plus courantes pour tirer parti des propriétés d'un contrôle.

Créer des procédures événementielles

Les contrôles placés sur une feuille sont réceptifs aux événements utilisateur qui les affectent. Vous pouvez ainsi créer des procédures dites événementielles, qui se déclencheront lorsque l'événement correspondant (un clic de souris, par exemple) sera repéré. Une procédure événementielle peut avoir des fonctions très variables, telles que vérifier la validité d'une information, modifier l'apparence de la feuille, ouvrir une autre feuille, fermer la feuille et passer les valeurs qu'elle contient à une procédure du module de code, etc.

Créer une procédure

Une procédure événementielle peut être générée de différentes façons. Quelle que soit la méthode choisie, les procédures événementielles d'une feuille ou des contrôles qui la composent répondent toujours à une même syntaxe :

```
Private Sub Contrôle_Événement ()  
    Instructions  
End Sub
```

L'instruction de déclaration de la procédure est toujours précédée de `Private`, car une procédure événementielle est par définition privée. Elle ne s'exécute que lorsque l'événement est repéré, et ne peut être appelé par une instruction située dans une procédure du projet.

Contrôle est le nom du contrôle auquel est attachée la procédure. Il s'agit de la valeur de sa propriété `Name`. *Événement* est l'événement qui déclenchera la procédure lorsqu'il affectera le contrôle. Il peut s'agir d'un clic de souris, d'une frappe de la touche Entrée ou encore d'une modification de la valeur du contrôle. Le nom du contrôle et celui de l'événement sont toujours séparés par un trait de soulignement.

Info

Dans le cas de procédures événementielles affectées à des feuilles, le nom de l'objet n'apparaît pas dans l'instruction de déclaration de la procédure. Le mot clé `UserForm` est dans ce cas employé pour identifier la feuille comme étant l'objet concerné par la procédure. L'instruction de déclaration d'une procédure événementielle de feuille répond à la syntaxe suivante :

```
Private Sub UserForm_Événement()
```

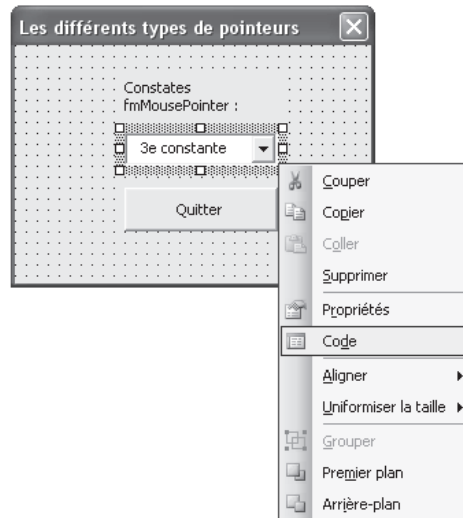
Vous pouvez écrire une procédure événementielle directement dans la fenêtre de Code de la feuille, en adoptant l'une des méthodes présentées plus loin dans ce chapitre. Pour ouvrir la fenêtre Code de la feuille, cliquez du bouton droit n'importe où sur celle-ci et choisissez la commande Code.

Les instructions de déclaration de la procédure peuvent cependant être générées automatiquement à partir de la feuille. Pour affecter une procédure événementielle à un contrôle, à partir de l'objet conteneur :

1. Double-cliquez sur le contrôle voulu,
ou cliquez du bouton droit et choisissez la commande Code (voir Figure 14.1),
ou sélectionnez le contrôle sur la feuille, puis choisissez la commande Code du menu Affichage.

Figure 14.1

Ouvrez la fenêtre Code à partir du contrôle auquel vous souhaitez affecter une procédure événementielle.

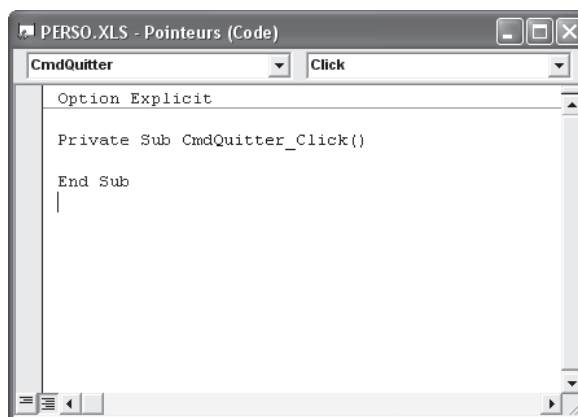


2. La fenêtre Code de la feuille s'ouvre. Les instructions de déclaration de la procédure événementielle du contrôle sont automatiquement générées et le curseur est placé entre celles-ci (voir Figure 14.2).

L'événement affecté à la procédure est l'événement par défaut du contrôle. Celui-ci varie d'un contrôle à l'autre. Il s'agit par exemple de l'événement Click (clic de souris) pour un contrôle CommandButton et de l'événement Change (modification de la valeur) pour un contrôle TextBox.

Figure 14.2

Les instructions de déclaration de la procédure sont automatiquement générées.

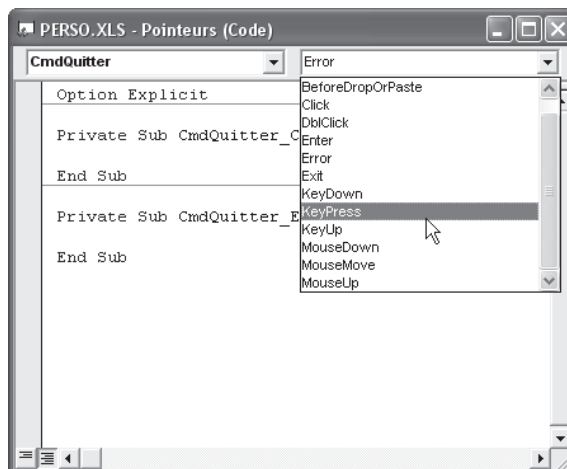


3. Si l'événement n'est pas celui pour lequel vous souhaitez créer une procédure, vous pouvez modifier au clavier le nom de l'événement généré, ou dérouler la zone de liste déroulante Procédure et y choisir l'événement voulu (voir Figure 14.3). Tous les événements pouvant être détectés sur le contrôle y sont listés.

Cette dernière solution génère les instructions de déclaration de la procédure événementielle sélectionnée. Supprimez alors les déclarations de procédure initialement générées.

Figure 14.3

La zone de liste déroulante Procédure liste l'ensemble des événements pouvant être détectés sur le contrôle affiché dans la zone Objet.



4. Saisissez le code de la procédure entre les instructions de déclaration.



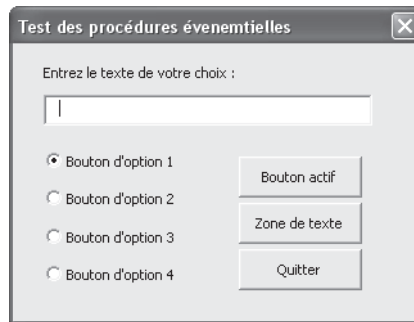
Si la procédure événementielle générée existe déjà, le curseur est placé sous son instruction de déclaration.

Pour réaliser l'exemple suivant, créez une feuille et placez-y un contrôle Label, un contrôle TextBox, quatre contrôles OptionButton et trois contrôles CommandButton, en définissant leurs propriétés décrites dans le tableau suivant.

Propriété	Valeur
Feuille	
Name	fmTestEvénements
Caption	Test des procédures événementielles
Contrôle Label	
Name	lbTexte
Value	Entrez le texte de votre choix
Contrôle TextBox	
Name	txtTexte
Value	
Contrôle OptionButton	
Name	De optOption1 à optOption4
Value	True pour l'un des boutons ; False pour les autres
Caption	De Bouton d'option 1 à Bouton d'option 4
Contrôle CommandButton	
Name	Respectivement cmdBouton, cmdTexte et cmdQuitter
Caption	Respectivement Bouton actif, Zone de texte et Quitter

Figure 14.4

La feuille Test des procédures événementielles réalisée.



Enfin, affectez les procédures événementielles suivantes au contrôle TextBox et aux trois contrôles CommandButton.

```
Private Sub txtTexte_AfterUpdate()
    MsgBox "Vous avez modifié la valeur de la zone de texte.", _
        vbOKOnly + vbInformation, "Événement AfterUpdate() détecté"
End Sub

Private Sub cmdBouton_Click()
    Dim BoutonActif As String
    If optOption1.Value = True Then
        BoutonActif = optOption1.Caption
    ElseIf optOption2.Value = True Then
        BoutonActif = optOption2.Caption
    ElseIf optOption3.Value = True Then
        BoutonActif = optOption3.Caption
    Else
        BoutonActif = optOption4.Caption
    End If
    MsgBox "Le bouton d'option sélectionné est le bouton libellé " & _
        BoutonActif, vbOKOnly + vbInformation, "Événement Click() détecté"
End Sub

Private Sub cmdTexte_Click()
    MsgBox "La valeur de la zone de texte est : " & txtTexte.Value, _
        vbOKOnly + vbInformation, "Événement Click() détecté"
End Sub

Private Sub cmdQuitter_Click()
    fmTestEvenements.Hide
End Sub
```

Sélectionnez la feuille dans la fenêtre UserForm et cliquez sur le bouton Exécuter de la barre d'outils Standard. La feuille s'affiche à l'écran. Testez le comportement de la feuille.

Lorsque les événements affectés à une procédure événementielle sont détectés, cette dernière s'exécute :

- La modification de la valeur de la zone de texte est interprétée comme l'événement `AfterUpdate` affectant le contrôle `txtTexte` lors du passage du focus à un autre contrôle. La procédure correspondante est exécutée et la boîte de dialogue de la Figure 14.5 s'affiche.
- Un clic sur le bouton libellé Bouton actif est détecté comme l'événement `Click` affectant le contrôle `cmdBouton`. La procédure correspondante est exécutée et la boîte de dialogue de la Figure 14.6 s'affiche.
- Un clic sur le bouton libellé Zone de texte est détecté comme l'événement `Click` affectant le contrôle `cmdTexte`. La procédure correspondante est exécutée et la boîte de dialogue de la Figure 14.7 s'affiche.
- Un clic sur le bouton libellé Quitter est détecté comme l'événement `Click` affectant le contrôle `cmdQuitter`. La procédure correspondante est exécutée et la fenêtre se ferme.

Figure 14.5

Déclenchement de la procédure `txtTexte_AfterUpdate`.

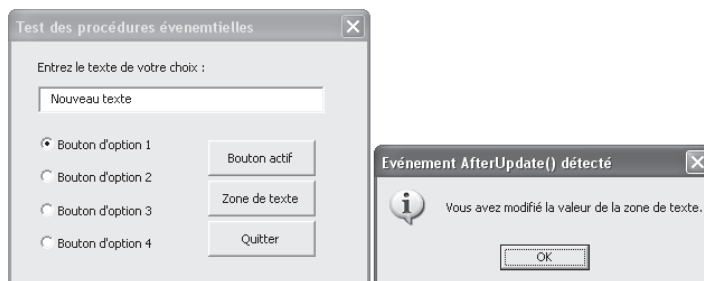


Figure 14.6

Déclenchement de la procédure `cmdBouton_Click`.

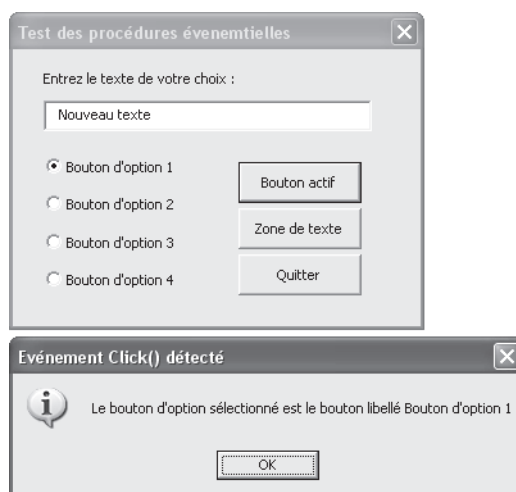
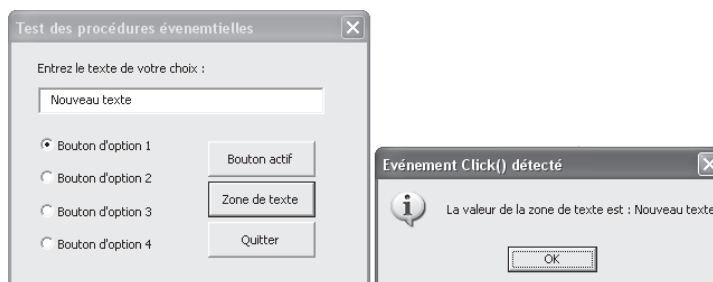


Figure 14.7

Déclenchement de la procédure `cmdTexte_Click`.



Les événements

Les événements sont nombreux et varient d'un contrôle à l'autre. Les événements gérés par un contrôle sont intimement liés à sa nature, c'est-à-dire à ses propriétés. Par exemple, les événements `Change`, `AfterUpdate` et `BeforeUpdate` sont gérés uniquement par les contrôles possédant une propriété `Value` (TextBox, par exemple), puisqu'ils se déclenchent lors de la modification de cette propriété. Ces événements ne sont pas gérés par les contrôles `CommandButton`, qui ne possèdent pas de propriété `Value`.

Astuce

Pour accéder à la liste des événements gérés par un type de contrôle, sélectionnez le contrôle en question sur une feuille et tapez sur `F1` pour ouvrir la rubrique d'aide qui lui est associée. Cliquez ensuite sur le lien `Événements` situé dans le haut de la rubrique d'aide. Une fenêtre listant les événements gérés par le contrôle s'affiche, permettant d'afficher la rubrique d'aide de l'événement de votre choix.

Cette section présente les événements gérés par les contrôles VBA. Certains, tels que les événements `Click`, `Change` ou `Initialize`, sont essentiels, et vous serez amené à les utiliser presque toujours lorsque vous développerez des feuilles VBA. Ils apparaissent en gras dans la liste. Il en est d'autres que vous n'utiliserez que très rarement, voire jamais.

- **Événement `AfterUpdate`**. Détection lorsque la valeur du contrôle est modifiée, au moment du passage du focus à un autre contrôle. Cet événement survient après l'événement `BeforeUpdate`.
- **Événement `BeforeDragOver`**. Détection lorsqu'un glisser-déplacer est effectué.
- **Événement `BeforeDropOrPaste`**. Détection lorsque des données sont déposées ou collées sur un objet. L'événement survient avant que l'action ne soit validée.
- **Événement `BeforeUpdate`**. Détection lorsque la valeur du contrôle est modifiée, au moment du passage du focus à un autre contrôle. L'événement `BeforeUpdate` survient

avant que la modification des données ne soit validée – et donc avant l'événement `AfterUpdate`. Cela permet de refuser la mise à jour effectuée. L'événement `BeforeUpdate` répond à la syntaxe suivante :

```
Private Sub Contrôle_BeforeUpdate(ByVal Annulation As MSForms.ReturnBoolean)
```

où `Annulation` est une variable de type `Boolean`.

Pour refuser la mise à jour, placez l'instruction `Annulation = True` dans la procédure.

L'utilisateur ne pourra alors pas passer le focus à un autre contrôle tant qu'une valeur valide n'aura pas été affectée au contrôle.

Dans l'exemple suivant, si l'utilisateur n'entre pas une valeur comprise entre 50 et 250 dans la zone de texte `txtTaille`, un message l'avertit que la valeur n'est pas valide, et le focus ne peut être passé à un autre contrôle tant qu'une valeur valide n'a pas été entrée.

```
1: Private Sub txtValeur_BeforeUpdate(ByVal Annulation _  
    As MSForms.ReturnBoolean)  
2:   If IsNumeric(txtValeur.Value) = False Then  
3:     MsgBox "Vous devez indiquer une valeur numérique"  
4:     Annulation = True  
5:   Else  
6:     If Not (txtValeur.Value >= 50 And txtValeur.Value <= 250) Then  
7:       MsgBox "Valeur non valide"  
8:       Annulation = True  
9:     End If  
10:  End If  
11: End Sub
```

L'instruction `If...End If` principale (des lignes 2 à 10) vérifie si la valeur entrée est une valeur numérique. Si tel n'est pas le cas (ligne 2), un message s'affiche (ligne 3), et la modification est refusée (ligne 4), interdisant à l'utilisateur de passer le focus à un autre contrôle. Si la valeur entrée est de type numérique (ligne 5), une instruction `If...End If` imbriquée (des lignes 6 à 9) vérifie que la valeur est comprise entre 50 et 250. Si tel n'est pas le cas, un message s'affiche (ligne 7), et la modification est refusée (ligne 8), interdisant à l'utilisateur de passer le focus à un autre contrôle.

- **Événement `Change`.** Détecté lors de la modification de la valeur (propriété `Value`) d'un contrôle. La modification peut être effectuée par l'utilisateur, comme générée par une instruction du code. Contrairement aux événements `AfterUpdate` et `BeforeUpdate` qui sont détectés lors du passage du focus à un autre objet, l'événement `Change` est détecté à chaque modification de la valeur. Ainsi, si vous affectez une procédure d'événement `Change` à un contrôle `TextBox`, celle-ci s'exécutera chaque fois que l'utilisateur entrera ou supprimera un caractère dans la zone de texte (s'il saisit le mot "Bonjour" dans la zone de texte, la procédure se déclenchera sept fois).

Il est fréquent d'utiliser une zone de texte pour afficher l'élément sélectionné par l'utilisateur dans une zone de liste. La procédure événementielle suivante met à jour la zone de texte txtSélliste chaque fois que l'utilisateur modifie la sélection dans la zone de liste ZoneDeListe :

```
Private Sub ZoneDeListe_Change()  
    txtSélliste.Value = ZoneDeListe.Value  
End Sub
```

- **Événement Click.** Détecté lorsque l'utilisateur clique sur un contrôle. Notez que l'événement Click est aussi détecté si l'équivalent clavier d'un clic de souris est effectué. C'est le cas si l'utilisateur frappe la touche Entrée alors qu'un bouton de commande a le focus, ou encore s'il frappe la barre d'espace alors qu'une case à cocher a le focus. L'événement Click survient après les événements MouseDown et MouseUp. Ainsi, lorsque vous cliquez sur un contrôle, trois événements sont successivement détectés :
 - MouseDown ;
 - MouseUp ;
 - MouseClick.
- **Événement dblClick.** Détecté lorsque l'utilisateur double-clique sur un contrôle. L'événement dblClick survient après l'événement Click. Lorsque vous double-cliquez sur un contrôle, quatre événements sont donc successivement détectés :
 - MouseDown ;
 - MouseUp ;
 - MouseClick ;
 - dblClick.
- **Événement DropButtonClick.** Détecté chaque fois que l'utilisateur déroule ou ferme la liste déroulante d'une zone de liste modifiable.
- **Événement Enter.** Détecté juste avant qu'un contrôle reçoive le focus. L'événement Enter est toujours lié à un événement Exit équivalent, qui affecte le contrôle qui avait le focus précédemment.
- **Événement Exit.** Détecté juste avant qu'un contrôle ne perde le focus. L'événement Exit répond à la syntaxe suivante :

```
Private Sub Contrôle_Exit(ByVal Annulation As MSForms.ReturnBoolean)
```

où Annulation est une variable de type Boolean.

Pour annuler le passage du focus, affectez la valeur True à l'argument Annulation dans une instruction de la procédure événementielle (voir l'exemple donné pour l'événement BeforeUpdate).

- **Événement Initialize.** Détecté lorsqu'une feuille est chargée. La procédure affectée à cet événement s'exécute avant l'affichage de la feuille et s'utilise selon la syntaxe suivante :

```
Private Sub UserForm_Initialize()  
    Instructions  
End Sub
```

Notez que, dans le cas de la procédure Initialize d'une feuille, le nom de la feuille n'est pas utilisé pour spécifier l'objet auquel est affectée la procédure. On utilise toujours le mot clé UserForm.

C'est une procédure essentielle de la programmation VBA. Elle permet en effet d'effectuer des réglages de la feuille, impossibles dans une fenêtre UserForm. Vous utiliserez, par exemple, une procédure événementielle Initialize pour affecter une liste d'éléments à un contrôle ListBox ou ComboBox de la feuille. Pour un exemple d'utilisation de la méthode Initialize, reportez-vous à la section "Contrôle *ComboBox*", plus loin dans ce chapitre.



Lorsque vous utilisez la méthode Hide pour masquer une feuille affichée à l'aide de la méthode Show, les ressources mémoire qu'elle occupe ne sont pas libérées. Si, par la suite, vous appelez de nouveau l'affichage de la feuille, l'événement Initialize ne sera pas reconnu, car la feuille ne sera pas chargée une nouvelle fois. Par conséquent, les différents contrôles de la feuille auront les mêmes valeurs que lorsqu'elle a été masquée. Pour libérer les ressources mémoire d'une feuille Userform, vous devez lui appliquer la méthode Unload, selon la syntaxe suivante :

Unload NomFeuille

- La méthode Load, elle, charge une feuille, déclenchant la procédure événementielle Initialize sans pour autant l'afficher. Notez que pour pouvoir manipuler une feuille par programmation, il faut que celle-ci soit chargée (à l'aide de la méthode Show ou de la méthode Load).
- **Événement KeyDown.** Détecté lorsqu'une touche du clavier est enfoncée. L'événement KeyDown répond à la syntaxe suivante :

```
Private Sub Contrôle_KeyUp(ByVal CodeTouche As MSForms.ReturnInteger, ByVal  
    EtatMaj As Integer)
```

où CodeTouche est une variable de type Integer renvoyant le code de touche de la touche frappée, et EtatMaj, une variable de type Integer renvoyant l'état des touches Maj, Ctrl et Alt. Lorsque aucune touche n'est enfoncée, EtatMaj prend la valeur 0. Si la touche

Maj est enfoncée, EtatMaj prend la valeur 1 ; s'il s'agit de la touche Ctrl, la valeur 2 est affectée à EtatMaj ; si la touche Alt est enfoncée, EtatMaj prend la valeur 4. Ces valeurs s'additionnent si plusieurs de ces touches sont enfoncées.

L'événement KeyDown alterne avec l'élément KeyPress. Lorsque l'utilisateur enfonce une touche, les événements KeyDown et KeyPress sont successivement détectés. S'il maintient la touche enfoncée, ces événements sont à nouveau détectés en série, le processus reflétant la périodicité d'insertion du caractère lorsque sa touche est maintenue enfoncée.

- **Événement KeyPress.** Cet événement est détecté lorsque l'utilisateur appuie sur une touche ANSI et intervient immédiatement après l'événement KeyDown. Si la touche ANSI est enfoncée, l'événement KeyPress est détecté en série, à chaque insertion d'un caractère. L'événement KeyPress répond à la syntaxe suivante :

```
Private Sub Contrôle_KeyPress(ByVal codeANSI As MSForms.ReturnInteger)
```

où *codeANSI* est le code ANSI de la touche enfoncée.

- **Événement KeyUp.** Détecté lorsqu'une touche est relâchée. L'événement KeyUp répond à la syntaxe suivante :

```
Private Sub Contrôle_KeyUp(ByVal CodeTouche As MSForms.ReturnInteger, ByVal  
EtatMaj As Integer)
```

Les valeurs affectées aux variables CodeTouche et EtatMaj sont les mêmes que pour l'événement KeyDown.

Pour visualiser le déroulement des événements KeyDown, KeyUp et KeyPress, placez un contrôle TextBox sur une feuille. Définissez sa propriété Name à txtTexte, puis placez le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
1: Private Sub TxtTexte_KeyDown(ByVal CodeTouche As MSForms.ReturnInteger, _  
  ByVal EtatMaj As Integer)  
2:   Dim EtatTouches As String      'Déclare la variable EtatTouches.  
3:   Select Case EtatMaj            'Interroge la valeur de la variable  
4:     Case 0                      'EtatMaj et affecte une valeur  
5:       EtatTouches = "Aucune"    'en conséquence à la variable  
6:     Case 1                      'EtatTouches  
7:       EtatTouches = "Maj"  
8:     Case 2  
9:       EtatTouches = "Ctrl"  
10:    Case 3  
11:      EtatTouches = "Maj et Ctrl"  
12:    Case 4  
13:      EtatTouches = "Alt"  
14:    Case 5  
15:      EtatTouches = "Maj et Alt"  
16:    Case 6  
17:      EtatTouches = "Ctrl et Alt"
```

```

18:      Case 7
19:          EtatTouches = "Maj, Ctrl et Alt"
20:      End Select
21:      Debug.Print ("Événement KeyDown détecté. Code touche = " & _
        CodeTouche & ". Touches enfoncées : " & EtatTouches)
22: End Sub

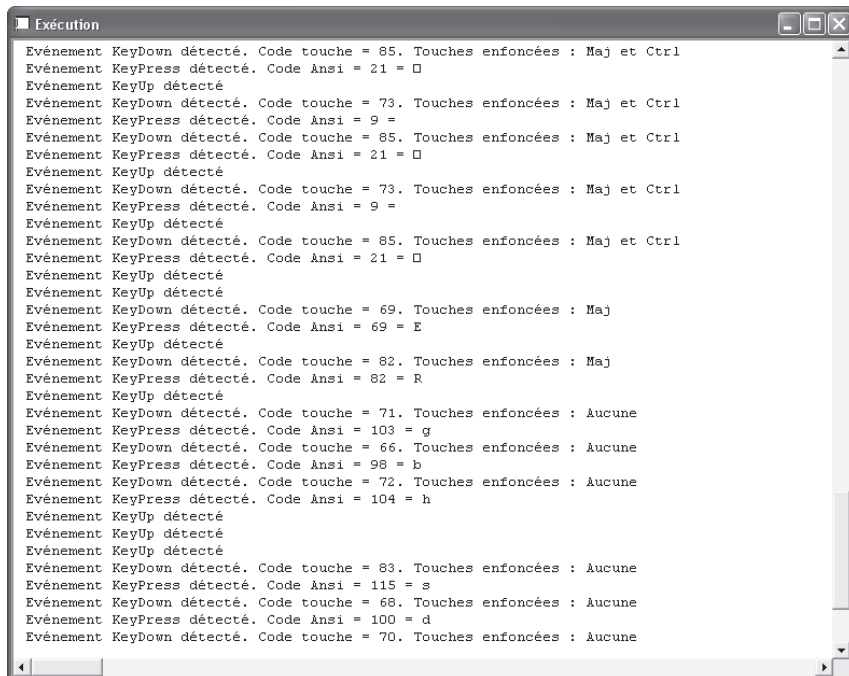
23: Private Sub txtTexte_KeyUp(ByVal CodeTouche As MSForms.ReturnInteger, _
        ByVal EtatMaj As Integer)
24:     Debug.Print ("Événement KeyUp détecté")
25: End Sub

26: Private Sub TxtTexte_Keypress(ByVal codeANSI As MSForms.ReturnInteger)
27:     Debug.Print ("Événement KeyPress détecté. Code Ansi = " & _
        codeANSI & " = " & Chr(codeANSI))
28: End Sub

```

Sélectionnez la feuille précédemment développée et cliquez sur le bouton Exécuter de la barre d'outils Standard. La feuille s'affiche. Testez différentes combinaisons clavier dans la zone de texte du contrôle, puis cliquez sur la case de fermeture de la feuille. Affichez la fenêtre Exécution (Ctrl+G). Le détail des événements détectés y est inscrit. Vous obtenez une fenêtre semblable à celle de la Figure 14.8.

Figure 14.8
*Les événements
clavier peuvent
être précisément
détectés par un
programme VBA.*



La première procédure est déclenchée lorsque l'événement `KeyDown` est détecté sur le contrôle `txtTexte`. La variable `EtatTouches` est déclarée de type `String` en ligne 2. Une instruction `Select Case...End Case` teste ensuite la valeur de la variable `EtatMaj` (lignes 3 à 20) renvoyée par l'événement `KeyDown`. Une chaîne de caractères est affectée à `EtatTouches` en fonction du résultat. L'instruction de la ligne 21 inscrit dans la fenêtre Exécution de la feuille les informations ainsi récoltées.

La deuxième procédure est déclenchée lorsque l'événement `KeyUp` est détecté sur le contrôle `txtTexte`. Une chaîne est alors insérée dans la fenêtre Exécution de la feuille (ligne 24).

La troisième procédure est déclenchée lorsque l'événement `KeyPress` est détecté sur le contrôle `txtTexte`. L'instruction de la ligne 27 insère dans la fenêtre Exécution une chaîne suivie de la valeur de la variable `codeANSI` renvoyée par `KeyPress`. Le caractère correspondant est ensuite inséré à l'aide de la fonction `Chr`.

- **Événement `MouseDown`.** Détecté lorsque l'utilisateur enfonce un bouton de souris. L'événement `MouseDown` répond à la syntaxe suivante :

```
Private Sub Contrôle_MouseDown(ByVal Bouton As fmButton, ByVal EtatMaj  
    As fmShiftState, ByVal X As Single, ByVal Y As Single)
```

où `Bouton` représente une valeur indiquant le bouton de souris qui a été enfoncé et `Maj`, une valeur reflétant l'état des touches `Maj`, `Ctrl` et `Alt`. `X` et `Y` indiquent respectivement les distances horizontale et verticale du curseur par rapport à l'angle supérieur gauche de la feuille, au moment du clic de souris.

La valeur affectée à `EtatMaj` est la même que pour les événements `KeyDown` et `KeyUp`. Si le bouton gauche est enfoncé, `Bouton` prend la valeur 1. La valeur 2 est affectée à `Bouton` si le bouton droit est enfoncé et la valeur 4, s'il s'agit du bouton central. Dans le cas où plusieurs boutons sont enfoncés, ces valeurs s'additionnent.

- **Événement `MouseMove`.** Détecté lorsque l'utilisateur déplace la souris. La syntaxe de l'événement `MouseMove` est la même que pour `MouseDown`.
- **Événement `MouseUp`.** Détecté lorsque l'utilisateur relâche un bouton de souris. La syntaxe de cet événement est la même que pour `MouseDown`.
- **Événement `SpinDown`.** Détecté lorsqu'un clic est effectué sur le bouton inférieur ou gauche d'un contrôle `SpinButton`. Lorsqu'un événement `SpinDown` est détecté, la propriété `Value` du contrôle est décrétementée de la valeur de la propriété `SmallChange`, à condition que la nouvelle valeur du contrôle soit supérieure à la valeur minimale du contrôle,

déterminée par sa propriété `Min`. Dans le cas contraire, le contrôle prend la valeur de la propriété `Min`.

- **Événement `SpinUp`.** Détecté quand on clique sur le bouton supérieur ou le bouton droit d'un contrôle `SpinButton`. Lorsqu'un événement `SpinUp` est détecté, la propriété `Value` du contrôle est incrémentée de la valeur de la propriété `SmallChange`, à condition que la nouvelle valeur du contrôle soit inférieure à la valeur maximale du contrôle, déterminée par sa propriété `Max`. Dans le cas contraire, le contrôle prend la valeur de la propriété `Max`.

Exemples d'exploitation des contrôles

Cette section présente les spécificités des contrôles qui sont à votre disposition dans la boîte à outils et les méthodes permettant de les exploiter pleinement.

Contrôle *Label*

Le texte d'un contrôle `Label` ne peut être modifié par l'utilisateur. Il peut cependant être utile de modifier la valeur d'un contrôle `Label` afin de créer une interface utilisateur dynamique. Il suffit pour cela de placer des instructions redéfinissant la valeur `Caption` du contrôle lorsqu'un événement spécifique survient. Vous pouvez ainsi placer un seul contrôle sur une feuille, destiné à remplir une fonction variable selon les informations entrées par l'utilisateur. C'est alors le contrôle `Label` identifiant ce contrôle qui permettra à l'utilisateur d'en distinguer la fonction.

Dans l'exemple suivant, un contrôle `Label` est affecté à une zone de texte, destinée à recevoir le nom d'un nouveau membre. Deux boutons d'option, respectivement libellés "Masculin" et "Féminin", permettent à l'utilisateur de spécifier le sexe du membre. L'information attendue de l'utilisateur varie selon le sexe du nouveau membre. Le bouton d'option actif par défaut est le bouton libellé "Masculin". Des procédures événementielles sont attachées à chacun des contrôles, afin que l'activation de l'un ou l'autre des boutons d'option modifie la propriété `Caption` du contrôle `Label`.

Pour réaliser cet exemple, créez une nouvelle feuille et placez-y un contrôle `Frame` contenant deux contrôles `OptionButton`, deux contrôles `TextBox`, associés tous deux à un contrôle `Label`, et deux contrôles `CommandButton`. Définissez ainsi leurs propriétés :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmLabel
Caption	Modification d'un contrôle Label
Contrôle Frame	
Name	frSexe
Caption	Sexe du nouveau membre
Contrôle OptionButton	
Name	optMasculin
Caption	Masculin
Value	True
Contrôle OptionButton	
Name	optFéminin
Caption	Féminin
Value	False
Contrôle Label	
Name	lbPrénom
Caption	Prénom
Contrôle TextBox	
Name	txtPrénom
Value	
Contrôle Label	
Name	lbNom
Caption	Nom
Contrôle TextBox	
Name	txtNom
Value	
Contrôle CommandButton	
Name	cmdOK
Caption	OK

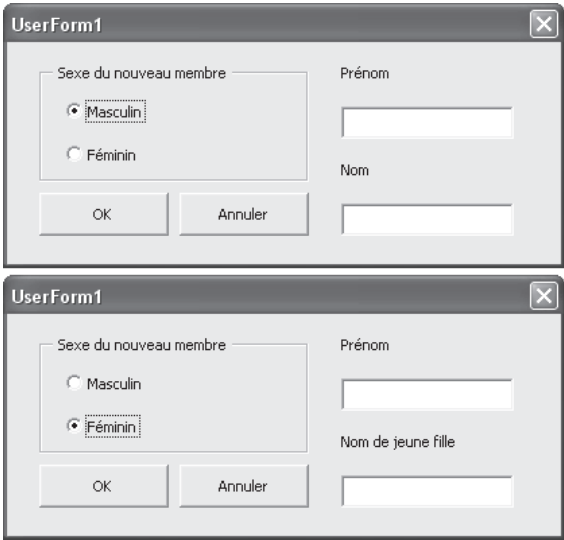
Propriété	Valeur
Contrôle CommandButton	
Name	cmdAnnuler
Caption	Annuler

Placez ensuite le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Sub optFéminin_Click()  
    lbNom.Caption = "Nom de jeune fille"  
End Sub  
  
Sub optMasculin_Click()  
    lbNom.Caption = "Nom"  
End Sub  
  
Sub cmdOK_Click()  
    fmLabel.Hide  
End Sub
```

Sélectionnez ensuite la feuille, puis cliquez sur le bouton Exécuter de la barre d’outils Standard. La feuille s’affiche à l’écran. Lorsque vous cliquez sur l’un des boutons d’option, le libellé du contrôle Label, lbNom, varie en fonction de l’option sélectionnée (voir Figure 14.9).

Figure 14.9
Faites varier le libellé d’un contrôle Label de façon à refléter l’information attendue.





*Lorsque vous écrivez du code modifiant la valeur **Caption** d'un contrôle **Label**, veillez à ce que la taille du contrôle permette l'affichage complet du nouveau libellé.*

Contrôle **TextBox**

Lorsque vous utilisez un contrôle **TextBox** destiné à recevoir une information bien spécifique, vérifiez que l'utilisateur entre une valeur valide, afin d'éviter que le programme ne génère une erreur. Par exemple, si vous créez une zone de texte dont la fonction est de recevoir une valeur représentant une somme à payer, assurez-vous que l'information entrée par l'utilisateur est bien une valeur numérique. Si les valeurs autorisées sont bien délimitées, vérifiez aussi que la valeur affectée au contrôle entre dans celles-ci.

Pour vérifier la valeur d'un contrôle **TextBox** au moment où l'utilisateur entre une information dans la zone d'édition du contrôle, attachez du code à l'événement **Change** de ce contrôle. Il est parfois aussi nécessaire de vérifier qu'une valeur a bien été entrée au moment de la validation des informations entrées par l'utilisateur.

Utilisez les fonctions **Visual Basic** permettant de déterminer les types de données pour vous assurer que les informations saisies par l'utilisateur sont valides.



*Pour un rappel des fonctions **Visual Basic** de contrôle de types de données, reportez-vous au Tableau 6.3.*



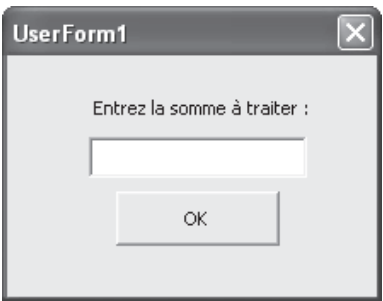
*Pour vérifier qu'une valeur est une chaîne de caractères, utilisez la fonction **IsNumeric** qui doit alors renvoyer la valeur **False**.*

Pour réaliser l'exemple suivant, créez une feuille présentant un contrôle **Label**, un contrôle **TextBox** et un contrôle **CommandButton** (voir Figure 14.10). Définissez ainsi leurs propriétés :

Propriété	Valeur
Feuille	
Name	fmMaFeuille
Caption	Vérification des données
Contrôle Label	
Name	lbValeur
Caption	Entrez la somme à traiter :

Propriété	Valeur
Contrôle TextBox	
Name	txtValeur
Value	
Contrôle CommandButton	
Name	cmdOK
Caption	OK

Figure 14.10
Votre feuille terminée.



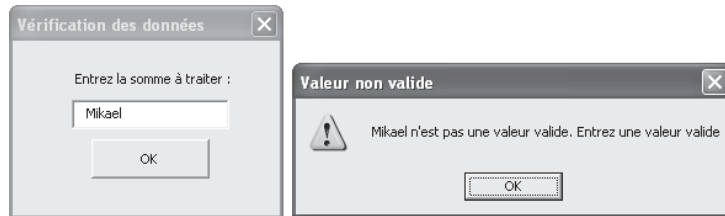
Placez ensuite le code suivant dans la zone Déclarations de la fenêtre Code de la feuille :

```
Private Sub txtValeur_AfterUpdate()  
    If IsNumeric(txtValeur.Value) = False Then  
        MsgBox txtValeur.Value & " n'est pas une valeur valide. Entrez une valeur  
valide", _  
            vbExclamation, "Valeur non valide"  
        txtValeur.Value = ""  
    End If  
End Sub  
  
Private Sub cmdOK_Click()  
    If txtValeur.Value = "" Then  
        MsgBox txtValeur.Value & "Vous devez entrer une valeur dans la zone de  
texte.", _  
            vbExclamation, "Valeur requise"  
    Else  
        fmMaFeuille.Hide  
    End If  
End Sub
```

La première procédure utilise la fonction `IsNumeric` dans une instruction conditionnelle `If...End If` pour s'assurer que la valeur entrée dans la zone de texte est une valeur numérique ou une chaîne vide. Si tel n'est pas le cas, une boîte de dialogue s'affiche à l'attention de l'utilisateur (voir Figure 14.11), et la propriété `Value` du contrôle est affectée à une chaîne vide.

Figure 14.11

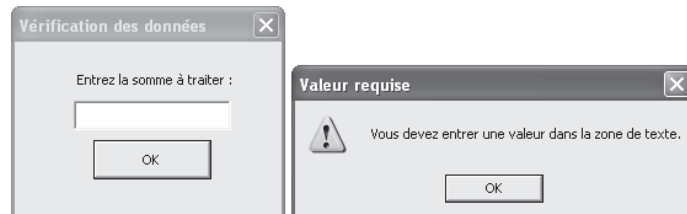
Une valeur non valide a été saisie dans la zone de texte.



La procédure `cmdOK_Clik` vérifie qu'une valeur a bien été entrée dans la zone de texte. Si tel n'est pas le cas, une boîte de dialogue s'affiche à l'attention de l'utilisateur (voir Figure 14.12). Si, au contraire, la zone de texte contient une valeur, alors la méthode `Hide` est appliquée à la feuille afin de la masquer.

Figure 14.12

Une valeur doit être saisie dans la zone de texte.



Notez que la procédure `Cmd_Clik` ne vérifie pas que la valeur de la zone de texte est valide (de type numérique), mais uniquement qu'elle est différente d'une chaîne vide. En effet, la procédure `txtValeur_Change` n'autorise pas la saisie d'une valeur autre que de type numérique dans la zone de texte.

Contrôle *ComboBox*

Les contrôles *ComboBox* sont parmi les plus utilisées dans les interfaces utilisateur de la plupart des logiciels. Ils présentent en effet l'avantage de permettre l'affichage d'un grand nombre d'options en n'occupant qu'un espace très limité sur la feuille.

Par défaut, un contrôle *ComboBox* se comporte à la manière d'un contrôle *TextBox*, c'est-à-dire que l'utilisateur est autorisé à saisir une valeur ne figurant pas parmi les options de la liste déroulante. Vous pouvez cependant déterminer un comportement analogue à celui d'un contrôle *ListBox*, c'est-à-dire n'autoriser que la sélection d'une valeur parmi les options de la liste déroulante, et interdire la saisie de nouvelles valeurs dans la zone d'édition du contrôle. Définissez pour cela la propriété *Style* du contrôle à *fmStyleDropDownList*. (voir la section dédiée à cette propriété, au chapitre précédent).

Ajout d'éléments à la liste d'un contrôle *ComboBox*

La méthode AddItem

La liste affectée à un contrôle *ComboBox* est généralement déterminée dans le code du programme, à l'aide de la méthode *AddItem*. Cette méthode s'utilise selon la syntaxe suivante :

```
Contrôle.AddItem(ElementAjoutéALaListe, Index)
```

où *Contrôle* est le nom (propriété *Name*) du contrôle *ComboBox*. *ElementAjoutéALaListe* est une chaîne de caractères représentant l'élément à ajouter et *Index*, la position de l'élément ajouté dans la liste. Le premier élément correspond à la valeur d'index 0, le second a la valeur d'index 1, etc. Ce dernier argument est généralement omis, l'élément ajouté est alors placé en dernière position dans la liste.



Lorsque l'argument Index est omis, l'argument ElementAjoutéALaListe peut être placé directement derrière AddItem, en ignorant les parenthèses.

Les instructions définissant les éléments de la liste d'un contrôle *ComboBox* sont généralement placées dans une procédure événementielle attachée à l'événement *Initialize* de la feuille. La liste est ainsi mise à jour au moment de l'affichage de la feuille.

Pour réaliser l'exemple suivant, placez un contrôle *Label*, un contrôle *ComboBox* et un contrôle *CommandButton* sur une feuille et définissez ainsi leurs propriétés :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmComboBox
Caption	Utilisation d'un contrôle ComboBox
Contrôle Label	
Name	lbComboBox
Caption	Sélection d'une valeur :
Contrôle ComboBox	
Name	cbComboBox
Value	
Contrôle CommandButton	
Name	cmdQuitter
Value	Quitter

Placez ensuite le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub UserForm_Initialize()
    Dim mavar
    For mavar = 1 To 10
        cbComboBox.AddItem "Element de liste " & mavar
    Next mavar
End Sub

Private Sub cmdQuitter_Click()
    fmComboBox.Hide
End Sub
```

Sélectionnez ensuite la feuille dans la fenêtre UserForm, puis cliquez sur le bouton Exécuter de la barre d'outils Standard. La feuille s'affiche. Déroulez la liste de la zone de texte modifiable pour visualiser les options disponibles (voir Figure 14.13). Pour fermer la feuille, cliquez sur le bouton Quitter.

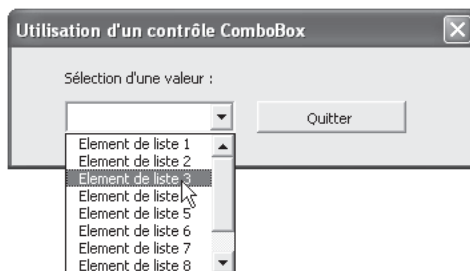
La procédure fmComboBox_Initialize est exécutée avant l'affichage de la feuille. La boucle For...Next s'exécute alors, ajoutant dix éléments à la liste du contrôle cbComboBox.



Utilisez la méthode RemoveItem pour supprimer des éléments d'une liste.

Figure 14.13

La liste est affectée au contrôle à l'affichage de la feuille.



La propriété RowSource

Les éléments de la liste peuvent aussi être affectés au contrôle grâce à la propriété RowSource. Cette propriété accepte pour valeur une chaîne représentant une cellule ou une plage de cellules. Cette possibilité est très pratique lorsqu'un projet est affecté à un document contenant des données variables. La liste reflète ainsi toujours les données de la feuille lorsque celles-ci sont modifiées.

Dans l'exemple suivant, une feuille VBA est créée afin de permettre à l'utilisateur d'entrer des informations sur les chiffres des différents vendeurs d'une société. Un contrôle ComboBox est placé sur la feuille afin de permettre à l'utilisateur de sélectionner le vendeur de son choix. Pour réaliser cet exemple, créez une feuille de calcul Excel et saisissez-y les données apparaissant à la Figure 14.14.

Figure 14.14

Les vendeurs sont répertoriés dans la colonne A.

	A	B	C
1	Vendeur	Région	Ville
2	Mary	Nord	Lille
3	du Château	Ile de France	Paris
4	Bidault	Ouest	Quimper
5	Des Neury	Sud Ouest	Biarritz
6	Gas quai	Sud Est	Marseille

Accédez ensuite à Visual Basic Editor (Alt+F11) et sélectionnez le projet attaché au nouveau document dans l'Explorateur de projet. Créez une feuille VBA et placez-y un contrôle Label, un contrôle ComboBox et un contrôle CommandButton. Définissez ainsi leurs propriétés :

Propriété	Valeur
Feuille	
Name	fmRowSource
Caption	Utilisation de la propriété RowSource
Contrôle Label1	
Name	lbComboBox
Caption	Sélectionnez un vendeur
Contrôle ComboBox	
Name	cbComboBox
Value	
RowSource	A2:A6
Contrôle CommandButton	
Name	cmdQuitter
Value	Quitter

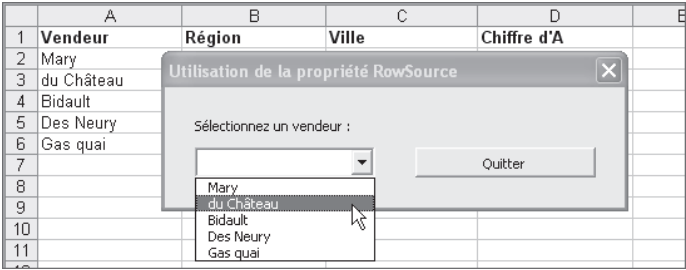
La définition de la propriété RowSource à A2:A6, affecte le contenu des cellules A2 à A6 de la feuille Excel à la liste du contrôle.

Placez ensuite le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub cmdQuitter_Click()  
    fmRowSource.Hide  
End Sub
```

Sélectionnez la feuille dans la fenêtre UserForm et cliquez sur le bouton Exécuter de la barre d’outils Standard. Déroulez la liste pour en visualiser le contenu (voir Figure 14.15), puis cliquez sur la case de fermeture de la fenêtre.

Figure 14.15
La liste déroulante reflète le contenu des cellules A2 à A6.



Le contrôle ComboBox reflète ainsi le contenu des cellules A2 à A6. Si vous ajoutez des vendeurs à la liste (en A7, puis A8, etc.), ceux-ci n'apparaîtront pas dans la liste déroulante du contrôle. Pour remédier à ce problème, il faut définir une procédure qui affecte à la propriété RowSource une plage de cellules variable, reflet des cellules contenant des informations. Redéfinissez la propriété RowSource du contrôle ComboBox à une chaîne vide, puis ajoutez la procédure suivante dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub UserForm_Initialize()
    Dim DerCell As String
    DerCell = Range("A1").End(xlDown).Address
    cbComboBox.RowSource = "A2:" & DerCell
End Sub
```

Cette procédure se déclenche à l'affichage de la feuille. La première instruction déclare la variable DerCell de type String. L'expression Range("A1").End(xlDown).Address renvoie l'adresse de la dernière cellule non vide sous la cellule A1. Cette adresse est affectée à la variable DerCell précédemment déclarée. Enfin, la propriété RowSource du contrôle cbComboBox est affectée à A2:DerCell, c'est-à-dire une plage de cellules allant de la cellule A2 à la dernière cellule de la colonne A contenant des informations.

Ajoutez de nouveaux noms à la liste des vendeurs, puis exécutez à nouveau la feuille. La liste déroulante affiche la totalité des vendeurs (voir Figure 14.16).

Figure 14.16

La liste affiche toujours la liste de tous les vendeurs.



Valeur renvoyée par un contrôle **ComboBox**

La valeur sélectionnée par l'utilisateur dans un contrôle ComboBox est définie par sa propriété Text ou sa propriété Value. La propriété Text renvoie une chaîne correspondant au libellé apparaissant dans la zone d'édition du contrôle. La valeur renvoyée par la propriété Value dépend de la propriété BoundColumn du contrôle. Si la propriété BoundColumn est définie à 0, la valeur de ListIndex est affectée à la propriété Value du contrôle. Autrement dit, la

propriété `Value` prendra la valeur 0 si le premier élément de la liste est sélectionné, 1 si le deuxième élément est sélectionné, etc., $n - 1$ si l'élément n est sélectionné.

Dans le cas d'un contrôle `ComboBox` à plusieurs colonnes, la valeur de la propriété `BoundColumn` détermine la colonne dont le contenu sera affecté à la propriété `Value`. Par exemple, si la propriété `BoundColumn` est définie à 3, la propriété `Value` du contrôle renverra une chaîne correspondant au libellé apparaissant dans la troisième colonne de la ligne sélectionnée. Pour plus d'informations sur la création de contrôles `ComboBox` à plusieurs colonnes, consultez l'Aide VBA.



Utilisez la propriété `ControlSource` pour affecter la valeur sélectionnée par l'utilisateur dans un contrôle `ComboBox` à une cellule d'une feuille de classeur. Par exemple, définir la propriété `ControlSource` d'un contrôle `ComboBox` à D4 insérera la valeur sélectionnée par l'utilisateur dans la liste dans la cellule D4, lors du passage du focus à un autre contrôle.

Contrôle *ListBox*

Les contrôles `ListBox` et `ComboBox` partagent nombre de propriétés (`RowSource`, `ControlSource`, `BoundColumn`, etc.) et de méthodes communes (`AddItem` et `RemoveItem`, par exemple). Pour affecter une liste à un contrôle `ListBox`, utilisez l'une des méthodes présentées ci-dessus pour le contrôle `ComboBox`.

Un contrôle `ListBox` peut n'autoriser que la sélection d'un élément de la liste, ou autoriser des sélections multiples. En outre, un contrôle `ListBox` autorisant des sélections multiples peut autoriser différentes méthodes de sélection.

Le type de sélection d'un contrôle `ListBox` est déterminé par sa propriété `MultiSelect`, qui accepte pour valeur l'une des constantes `fmMultiSelect` :

- **`fmMultiSelectSingle`** (par défaut). Un seul élément peut être sélectionné.
- **`fmMultiSelectExtended`**. Plusieurs éléments peuvent être sélectionnés avec les touches Maj ou Ctrl. Un clic sur un élément de la liste, sans qu'aucune de ces touches ne soit enfoncée, désélectionne tous les éléments pour sélectionner l'élément cliqué.
- **`fmMultiSelectMulti`**. Plusieurs éléments peuvent être sélectionnés ou désélectionnés par de simples clics. Un clic sur un élément de la liste sélectionne cet élément (ou le désélectionne s'il est déjà sélectionné) sans modifier l'état des autres éléments de la liste.

Figure 14.17

Une zone de liste n'autorisant qu'une sélection unique.

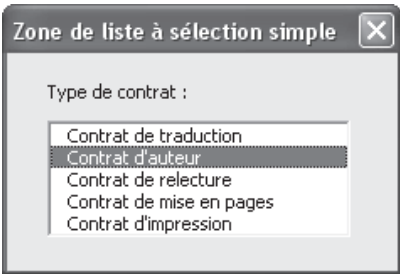
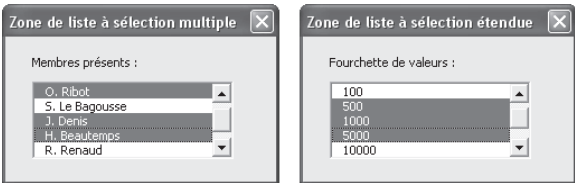


Figure 14.18

Une zone de liste autorisant des sélections multiples.



Valeur renvoyée par un contrôle *ListBox*

À l'instar des contrôles *ComboBox*, la valeur d'un contrôle *ListBox* à sélection unique peut être renvoyée par sa propriété *Value* ou *Text*. Dans le cas d'un contrôle autorisant des sélections multiples, ces propriétés renvoient une chaîne vide. Pour déterminer les éléments sélectionnés d'une zone de liste autorisant de multiples sélections, vous devez interroger l'état de chacune des options de la liste. Utilisez pour cela la propriété *Selected* du contrôle *ListBox* selon la syntaxe suivante :

*Contrôle**ListBox*.*Selected*(*IndexElement*) = *ValeurBooléenne*

où *Contrôle**ListBox* est le nom (propriété *Name*) du contrôle et *IndexElement*, sa position dans la liste (propriété *ListIndex*). La *ValeurBooléenne* renvoyée est *True* si l'élément est sélectionné et *False*, dans le cas contraire.

L'exemple suivant inscrit les éléments sélectionnés dans une zone de liste dans la fenêtre Exécution. Pour réaliser cet exemple, placez un contrôle *Label*, un contrôle *ListBox* et un contrôle *CommandButton* sur une feuille et définissez ainsi leurs propriétés :

Propriété	Valeur
Feuille	
Name	fmListBox
Caption	Valeurs sélectionnées dans la liste

<i>Propriété</i>	<i>Valeur</i>
Contrôle Label	
Name	lbMembres
Caption	Membres présents
Contrôle ListBox	
Name	cbMembres
Value	
MultiSelect	fmMultiSelectMulti
Contrôle CommandButton	
Name	cmdValider
Caption	Valider

Placez ensuite le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub UserForm_Initialize()
    cbMembres.AddItem "Bidault"
    cbMembres.AddItem "Deschamps"
    cbMembres.AddItem "Kervarrec"
    cbMembres.AddItem "Goraguer"
    cbMembres.AddItem "Lemaire"
    cbMembres.AddItem "Leroux"
    cbMembres.AddItem "Martin"
    cbMembres.AddItem "Opéra"
    cbMembres.AddItem "Otello"
End Sub

Private Sub cmdValider_Click()
    Dim compteur As Single
    For compteur = 0 To (cbMembres.ListCount - 1)
        If cbMembres.Selected(compteur) = True Then
            Debug.Print cbMembres.List(compteur)
        End If
    Next compteur
    fmListBox.Hide
End Sub
```

La première procédure affecte une liste au contrôle ListBox, à l’affichage de la feuille. La procédure cmdValider_Click se déclenche lorsque l’utilisateur clique sur le bouton Valider de la feuille. Elle utilise les propriétés du contrôle ListBox suivantes :

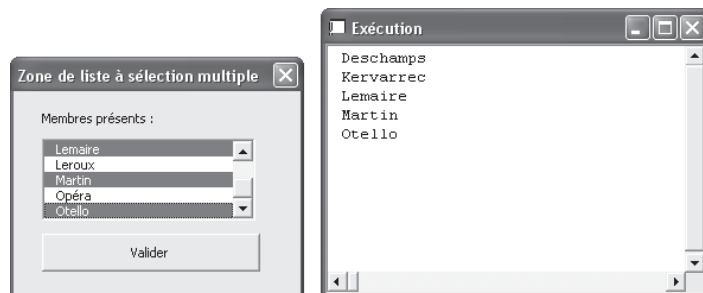
- **ListCount**. Renvoie le nombre d'éléments de la liste.
- **Selected(index)**. Renvoie une valeur booléenne indiquant si l'élément d'index `index` est sélectionné.
- **List(index)**. Renvoie la chaîne de caractères correspondant à l'élément d'index `index`.

Une boucle `For...Next` est utilisée pour interroger la valeur de chacun des éléments de la liste. Le premier élément de la liste ayant pour index 0, l'index du dernier élément est égal au nombre total d'éléments de la liste, moins 1. La boucle s'exécute donc sur un compteur allant de 0 au nombre d'éléments, moins 1. À chaque passage de la boucle, l'état de l'élément d'index `compteur` est interrogé. Si l'élément interrogé est sélectionné, l'instruction `Debug.Print` affiche le nom de l'élément dans la fenêtre Exécution de Visual Basic Editor. Enfin, la méthode `Hide` est appliquée à la feuille afin de le fermer.

Sélectionnez la feuille dans la fenêtre UserForm et cliquez sur le bouton Exécuter de la barre d'outils Standard. Sélectionnez les éléments de votre choix dans la liste, puis cliquez sur le bouton Valider. La feuille se ferme. Affichez la fenêtre Exécution de Visual Basic Editor (Ctrl+G). Les éléments sélectionnés dans la liste y sont inscrits (voir Figure 14.19).

Figure 14.19

Interrogez un à un les éléments d'une liste à sélection multiple, afin de déterminer les éléments sélectionnés par l'utilisateur.



Contrôles *CheckBox* et *OptionButton*

Le contrôle `CheckBox` est d'une utilisation simple. Pensez à utiliser les propriétés `Enabled` et `Visible` pour modifier l'accessibilité du contrôle, en fonction des informations entrées par l'utilisateur. Vous pouvez aussi modifier la propriété `Caption` d'un contrôle `CheckBox`, afin d'en faire varier la fonction. Pour un exemple d'utilisation de ce contrôle, reportez-vous à la section consacrée à la propriété `Enabled`, plus haut au chapitre précédent.

Pour associer des contrôles `OptionButton`, vous pouvez les placer sur un même contrôle `Frame` ou leur affecter une même propriété `GroupName`. Reportez-vous à la section dédiée à la propriété `GroupName`. Pour un exemple d'exploitation de contrôles `OptionButton`, reportez-vous à la section consacrée à la propriété `Visible`.

Contrôle *ScrollBar*

L'exemple suivant utilise un contrôle `TextBox` et un contrôle `ScrollBar` pour permettre à l'utilisateur de rechercher les années bissextiles comprises entre l'an 0 et l'an 3000. Afin d'accélérer la recherche d'une année bissextile, l'utilisateur pourra se déplacer par cent années en cliquant dans la barre de défilement. Pour réaliser cet exemple, placez un contrôle `TextBox` et un contrôle `ScrollBar` sur une feuille et définissez leurs propriétés ainsi :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmAnnéesBissextiles
Caption	Les années bissextiles
Contrôle TextBox	
Name	txtAnnée
Value	L'an 2000 est une année bissextile
Locked	True
Contrôle ScrollBar	
Name	scrAnnée
Value	2000
Min	0
Max	3000
SmallChange	4
LargeChange	100

Ajoutez ensuite le code suivant dans la section Déclarations de la fenêtre Code de la feuille :

```
Private Sub scrAnnée_Change()
    Dim varAnnée As Single
    varAnnée = scrAnnée.Value
    txtAnnée.Value = "L'an " & varAnnée & " est une année bissextile."
End Sub
```

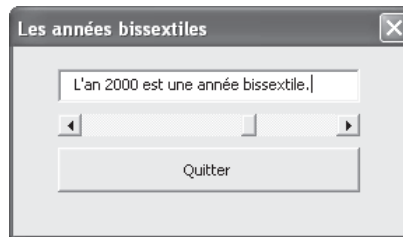
La procédure `scrAnnée_Change` sera déclenchée chaque fois que l'utilisateur modifiera l'emplacement du curseur sur le contrôle `ScrollBar` nommé "scrAnnée". Cette procédure déclare la variable `varAnnée` et lui affecte la valeur de la propriété `Value` de `scrAnnée`. La propriété `Value` du contrôle `TextBox` `txtAnnée` est ensuite définie, et la zone d'édition

affiche la chaîne "L'an *varAnnée* est une année bissextile.", où *varAnnée* est la valeur de la variable *varAnnée*.

Sélectionnez la feuille et cliquez sur le bouton Exécuter de la barre d'outils Standard afin d'exécuter la feuille. La feuille présentée à la Figure 14.20 s'affiche. La barre de défilement permet de modifier l'année affichée dans la zone de texte. Lorsque vous cliquez sur l'une des flèches de défilement, la valeur du contrôle est incrémentée ou décrémentée de 4 (la valeur de *SmallChange*). Lorsque vous cliquez dans la barre, entre le curseur et une des flèches de défilement, la valeur du contrôle est incrémentée ou décrémentée de 100 (la valeur de *LargeChange*).

Figure 14.20

Pour atteindre une année éloignée de la valeur actuelle, déplacez-vous 100 ans par 100 ans, en cliquant à l'intérieur de la barre plutôt que sur les flèches de défilement.



Ce programme fonctionne bien tant que l'utilisateur déplace le curseur en cliquant sur l'une des flèches de la barre de défilement, ou à l'intérieur de celle-ci. Mais si l'utilisateur déplace le curseur en faisant glisser le curseur sur la barre de défilement, la valeur du contrôle *scrAnnée* n'est plus maîtrisée par le programme. Une valeur ne représentant pas une année bissextile peut alors être sélectionnée et affichée dans la barre de défilement.

Partant du constat que toute année bissextile divisée par 4 est égale à un nombre entier, et que tout nombre multiple de 4 correspond à une année bissextile, vous pouvez exploiter cette condition pour vérifier que la valeur du contrôle correspond à une année bissextile. La condition nécessaire et suffisante pour que la valeur du contrôle *scrAnnée* corresponde à une année bissextile est donc :

```
Int(scrAnnée.Value / 4) = scrAnnée.Value / 4
```

La fonction Visual Basic *Int* renvoie la valeur entière du nombre spécifié entre parenthèses. La condition posée ici est que la valeur renvoyée par la division de *scrAnnée* par 4 est égale à la valeur entière de cette division.

Modifiez la procédure *scrAnnée_Change* de la façon suivante :

```
Private Sub scrAnnée_Change()  
    While Not Int(scrAnnée.Value / 4) = scrAnnée.Value / 4  
        scrAnnée.Value = scrAnnée.Value - 1  
    Wend
```

```
Dim varAnnée As Single
varAnnée = scrAnnée.Value
txtAnnée.Value = "L'an " & varAnnée & " est une année bissextile."
End Sub
```

La boucle While...Wend décrémente la valeur de scrAnnée tant que la condition posée n'est pas respectée (While Not) – la boucle s'exécute donc au maximum trois fois. Ainsi, si l'utilisateur opère un glisser-déplacer du curseur de défilement, nous sommes assurés que la valeur de scrAnnée sera modifiée si nécessaire, afin de correspondre à une année bissextile.

Contrôle *SpinButton*

Dans l'exemple suivant, un contrôle SpinButton est placé à côté d'un contrôle TextBox, et permet de sélectionner sept valeurs. Un jour de la semaine est affecté à chacune des sept valeurs possibles et affiché dans la zone de texte. Pour réaliser cet exemple, placez un contrôle TextBox et un contrôle SpinButton, une feuille et attribuez-leur les propriétés suivantes :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmJour
Caption	Choix du jour
Contrôle TextBox	
Name	txtJour
Value	Lundi
Locked	True
Contrôle SpinButton	
Name	spbJour
Value	7
Min	1
Max	7
SmallChange	1

Ouvrez ensuite la fenêtre de code de la feuille et placez le code suivant (sans la numérotation des lignes) dans la section Déclarations de la fenêtre Code de la feuille :

```
1: Private Sub spbJour_Change()  
2:   Dim varBoutonToupie As Single  
3:   varBoutonToupie = spbJour.Value  
4:   txtJour.Value = QuelJour(varBoutonToupie)  
5: End Sub  
  
6: Private Function QuelJour(varBoutonToupie)  
7:   Select Case varBoutonToupie  
8:     Case 1  
9:       QuelJour = "Dimanche"  
10:    Case 2  
11:      QuelJour = "Samedi"  
12:    Case 3  
13:      QuelJour = "Vendredi"  
14:    Case 4  
15:      QuelJour = "Jeudi"  
16:    Case 5  
17:      QuelJour = "Mercredi"  
18:    Case 6  
19:      QuelJour = "Mardi"  
20:    Case 7  
21:      QuelJour = "Lundi"  
22:   End Select  
23: End Function
```

Sélectionnez la feuille et cliquez sur le bouton Exécuter de la barre d'outils Standard afin d'exécuter la feuille. La feuille présentée à la Figure 14.21 s'affiche. Le bouton toupie permet de modifier le jour affiché dans la zone de texte. Lorsque Dimanche s'affiche dans la zone de texte, la flèche de défilement Bas du bouton toupie est sans effet. Lorsque Lundi s'affiche dans la zone de texte, la flèche de défilement Haut est sans effet.

Figure 14.21

Un clic sur une flèche de défilement du contrôle augmente ou diminue sa valeur de 1 et affiche le jour correspondant.



La procédure `spbJour_Change` se déclenche chaque fois que la valeur du contrôle `spbJour` est modifiée, c'est-à-dire chaque fois que l'utilisateur clique sur l'une des flèches de défilement du bouton toupie. La variable `varBoutonToupie` est alors déclarée (ligne 2) et la valeur

du contrôle `SpinButton` `spbJour` lui est affectée. À la ligne 4, la propriété `Value` du contrôle `TextBox` `txtJour` est affectée à l'expression `QuelJour(VarBoutonToupie)`, qui appelle la fonction `QuelJour` en lui passant l'argument `varBoutonToupie`.

La fonction `QuelJour` détermine le texte à afficher dans la zone d'édition du contrôle. Des lignes 7 à 22, une instruction de contrôle `Select Case...End Select` est utilisée pour interroger la valeur de `varBoutonToupie`. Pour chacune des valeurs possibles pour `varBoutonToupie`, une chaîne de caractères est affectée à la fonction.

La fonction `QuelJour` terminée, la procédure appelante reprend la main, et l'instruction de la ligne 4 peut s'exécuter. La valeur de la fonction `QuelJour` est alors affichée dans la zone de texte.

Nous allons maintenant modifier ce programme afin que les flèches de défilement Haut et Bas du bouton toupie permettent de passer de Lundi à Dimanche, et inversement. Une technique courante pour qu'un bouton toupie génère des valeurs en boucle consiste à autoriser une valeur minimale et une valeur maximale de plus pour le contrôle, qui ne seront pas visibles pour l'utilisateur, mais exploitées par une instruction conditionnelle pour créer la boucle.

Définissez la propriété `Min` du contrôle `SpinButton` à 0 et sa propriété `Max`, à 8. Modifiez la procédure `spbJour_Change` de la façon suivante :

```
1: Private Sub spbJour_Change()
2:   If spbJour.Value = 0 Then
3:     spbJour.Value = 7
4:   ElseIf spbJour.Value = 8 Then
5:     spbJour.Value = 1
6:   End If
7:   Dim varBoutonToupie As Single
8:   varBoutonToupie = spbJour.Value
9:   txtJour.Value = QuelJour(varBoutonToupie)
10: End Sub
```



Lorsque la propriété `Value` d'un contrôle `ScrollBar` ou `SpinButton` est égale à sa propriété `Max` ou `Min`, la redéfinition de la valeur de l'une de ces propriétés dans la fenêtre `Propriétés` entraîne aussi la redéfinition de la propriété `Value`. Dans le cas présent, l'affectation de la valeur 8 à la propriété `Max` redéfinira aussi la propriété `Value` du contrôle à 8. Vous devez par conséquent redéfinir à 7 la propriété `Value`, après avoir modifié la valeur de la propriété `Max`.

La structure conditionnelle ajoutée interroge la valeur du contrôle `spbJour`. Si l'utilisateur clique sur la flèche de défilement Haut du bouton toupie alors que sa valeur est 1 (la zone

de texte affiche Lundi), sa valeur passe à 0 (Min) et la structure conditionnelle (lignes 2 et 3) la redéfinit à 7 (Max moins 1). Si l'utilisateur clique sur la flèche de défilement Bas du bouton toupie alors que sa valeur est 7 (la zone de texte affiche Dimanche), sa valeur passe à 8 (Max) et la structure conditionnelle (lignes 4 et 5) la redéfinit à 1 (Min plus 1).



La procédure conditionnelle gérant les valeurs en boucle du bouton toupie peut tout aussi bien s'écrire ainsi :

```
If spbJour.Value = spbJour.Min Then  
    spbJour.Value = spbJour.Max - 1  
ElseIf spbJour.Value = spbJour.Max Then  
    spbJour.Value = spbJour.Min + 1  
End If
```

Exploiter les informations d'une feuille VBA

Pour exploiter les informations d'une feuille VBA, vous devez passer les informations que contient cette feuille à une procédure d'un module de code. Il vous suffit d'appeler la procédure qui récupérera et traitera ces informations en spécifiant les valeurs des contrôles comme arguments passés.



Les appels de procédure et le passage d'arguments à la procédure appelée ont été traités au Chapitre 5.

Lorsque vous souhaitez développer une interface utilisateur pour simplifier une tâche, procédez selon les étapes suivantes :

1. Développez votre feuille. Déterminez les propriétés des différents contrôles et écrivez les procédures événementielles spécifiques aux contrôles.
2. Dans un module de code, écrivez une procédure qui affichera la feuille (NomFeuille.Show). Vous pourrez éventuellement affecter par la suite cette procédure à un bouton de barre d'outils ou à une commande de menu (voir Chapitre 11), de façon à simplifier l'exécution du programme.
3. Votre feuille contiendra probablement un bouton de validation (OK). Créez une procédure événementielle pour l'événement Click de ce bouton. Cette procédure devra :
 - vérifier que les informations entrées par l'utilisateur sont valides ;
 - masquer la feuille (Me.Hide) ;
 - appeler la procédure qui traitera les informations.

La procédure événementielle suivante appelle la procédure `ValidationConditions`, en lui passant pour arguments les valeurs des contrôles `TxtVolume`, `TxtPrixTrad`, `TxtPrixAudio`, `chkDefinitif` et `chkImprimer`.

```
Private Sub CmdOK_Click()  
    'Instructions vérifiant la validité des données  
    Me.Hide  
    Call ValidationConditions(TxtVolume.Value, TxtPrixTrad.Value, _  
        TxtPrixAudio.Value, ChkDefinitif.Value, ChkImprimer.Value)  
End Sub
```

La procédure `ValidationConditions` devra se trouver dans un module de code standard. Son instruction de déclaration comprendra les arguments correspondant aux valeurs passées par la procédure appelante. Elle pourra par exemple se présenter comme suit :

```
Sub ValidationConditions(Volume, PrixTrad, Prix, DefinitifOuNon, ImprimerOuNon)  
    'Instructions  
End Sub
```



Si vous devez recueillir de nombreuses informations, créez plusieurs feuilles VBA et structurez-les de la façon suivante :

- 1. Affichage de la première feuille.*
 - 2. Stockage des données de la feuille dans des variables d'un module de code.*
 - 3. Masquage de la feuille*
 - 4. Affichage de la seconde feuille et retour au point 2.*
- Etc.*
- N. Appel de la procédure en charge de traiter les données entrées dans les différentes feuilles.*

IV

Notions avancées de la programmation Excel

CHAPITRE 15. Programmer des événements Excel

CHAPITRE 16. Protéger et authentifier des projets VBA

CHAPITRE 17. Exemple complet d'application Excel



15

Programmer des événements Excel

Au sommaire de ce chapitre

- L'objet *Application*
- L'objet *ThisWorkbook*
- L'objet *Worksheet*

Vous avez découvert, au cours de cet ouvrage, l'environnement de Visual Basic Editor, les techniques de programmation en VBA et les outils d'aide au développement de projet. Ce chapitre vous apprendra à gérer précisément les événements utilisateur susceptibles d'affecter les objets Excel au cours d'une utilisation classique du logiciel. Vous verrez qu'il est possible de détecter les actions de l'utilisateur sur un classeur et de créer des procédures affectées à ces événements.

Au même titre que les procédures événementielles créées pour les contrôles ActiveX que vous placez sur une feuille UserForm ou sur une feuille de calcul Excel, les classeurs et les feuilles sont des objets auxquels vous pouvez affecter des procédures événementielles. Vous pouvez ainsi gérer des événements tels que la création d'un nouveau classeur, la modification d'une cellule, l'activation d'une feuille, etc.

L'objet *Application*

Au sommet du modèle d'objets d'Excel se trouve l'objet *Application*. Il représente l'ensemble de l'application et est donc l'objet conteneur de tous les objets qui la composent. L'objet *Application* d'Excel est particulièrement intéressant pour le développeur. Il intègre en effet la gestion d'événements de niveau application, susceptibles d'intervenir lors d'une utilisation courante d'Excel : création d'un nouveau classeur, ouverture ou fermeture d'un classeur, etc. Vous pouvez ainsi créer des procédures événementielles destinées à gérer lesdits événements.

Les sections suivantes vous indiquent la procédure à suivre pour créer des procédures événementielles pour l'objet *Application*.

Déclaration et instanciation de l'objet *Application*

Les procédures destinées à gérer les événements utilisateur de niveau application ne peuvent être écrites que dans un module de classe. Pour écrire des procédures VBA destinées à gérer des événements de niveau application, vous devez créer un module de classe et y déclarer une variable objet de type *Application* à l'aide du mot clé *WithEvents*. Ce dernier indique que l'objet ainsi créé gèrera les événements.

1. Affichez l'Explorateur de projet. Cliquez du bouton droit sur l'un des éléments du projet et, dans le menu contextuel qui s'affiche, choisissez *Insertion > Module de classe*.
2. Par défaut, le nouveau module est nommé *Class1* (*Class2* si le module *Class1* existe déjà, etc.). Sélectionnez-le puis appuyez sur la touche F4. Dans la fenêtre *Propriétés*, affectez un nom représentatif au module (ici, *ModuleGestionEvt*).
3. Ouvrez la fenêtre *Code* du module et saisissez-y l'instruction suivante :

```
Public WithEvents MaVarApplication As Application
```



Le mot clé WithEvents n'est valide que dans un module de classe.

Notre projet intègre maintenant un module de classe dans lequel nous avons défini un objet `MaVarApplication` de type `Application`, capable de gérer les événements. Pour pouvoir l'utiliser, nous devons maintenant créer une instance de cet objet et l'affecter à l'objet `Application` d'Excel. Cette procédure peut se trouver dans n'importe quel module de code.

Supposons ici que vous souhaitez que la gestion des événements de niveau application soit active dès l'ouverture d'Excel. Il vous suffit d'écrire l'instruction d'instanciation de la variable dans une procédure événementielle, affectée à l'ouverture du classeur `PERSO-NAL.XLSB`.

Procédez comme suit :

1. Dans l'Explorateur de projet, double-cliquez sur l'objet `ThisWorkbook` du dossier `Microsoft Excel Objets`. Sa fenêtre `Code` s'affiche. Dans la section `Déclaration` de la fenêtre `Code`, placez l'instruction de déclaration suivante :

```
Dim MonInstance As New ModuleGestionEvt
```

où `ModuleGestionEvt` est le nom affecté au module de classe créé dans la section précédente. Nous avons ainsi créé une instance de la classe `ModuleGestionEvt`.

2. Dans la zone `Objet` de la fenêtre `Code`, sélectionnez `Workbook`. Les instructions d'encadrement d'une procédure événementielle de type `Open` sont automatiquement créées. Placez entre celles-ci l'instruction d'affectation de la variable `MonInstance`. La procédure doit se présenter comme suit :

```
Private Sub Workbook_Open()  
    Set MonInstance.MaVarApplication = Application  
End Sub
```

L'instruction d'affectation de cette procédure affecte l'objet `Application` à la propriété `MaVarApplication` de l'objet `MonInstance`.

Vous pouvez maintenant créer des procédures événementielles de niveau application.

Création de procédures événementielles de niveau application

Les procédures événementielles de niveau application se trouveront dans le module de classe créé dans la section précédente. La création de procédures événementielles de niveau application s'effectue de la même façon que les procédures événementielles de contrôles `ActiveX`.

1. Ouvrez la fenêtre Code du module de classe ModuleGestionEvt.
2. Dans la zone Objet, choisissez la variable MaVarApplication. Les instructions d'encadrement d'une procédure événementielle de type NewWorkbook (événement nouveau classeur) sont automatiquement insérées. Placez une instruction d'affichage de boîte de dialogue à l'aide de la fonction MsgBox :

```
Private Sub MaVarApplication_NewWorkbook(ByVal Wb As Excel.Workbook)
    MsgBox «L'utilisateur a créé un nouveau classeur», vbOKOnly + vbInformation
End Sub
```

L'argument Wb de l'instruction de déclaration de la procédure événementielle correspond à l'objet Workbook créé. Vous pouvez utiliser cette variable dans votre procédure pour interroger et manipuler le classeur créé. L'instruction suivante affiche une boîte de dialogue dans laquelle est mentionné le nombre de feuilles du classeur :

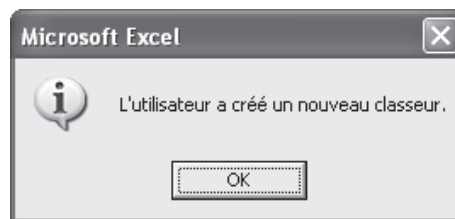
```
MsgBox «Le classeur contient « & Wb.Sheets.Count & « feuilles.»
```

3. Enregistrez le projet et quittez Excel.
4. Relancez Excel et créez un nouveau classeur. La boîte de dialogue présentée à la Figure 15.1 s'affiche.

La reconnaissance de l'événement NewWorkbook permet de créer des procédures destinées à aider l'utilisateur dans ses tâches courantes. Vous pouvez par exemple afficher une feuille UserForm contenant une liste d'options de classeurs. L'utilisateur sera alors invité à indiquer le type de classeur qu'il souhaite réaliser (Ventes du mois, Représentants, (Autre), etc.). La procédure appelée pourra insérer les données essentielles, créer le nombre de feuilles voulues, enregistrer le classeur dans le bon dossier, etc.

Figure 15.1

Une procédure peut être affectée à la création d'un nouveau classeur.



La zone de liste Procédure recense les événements gérés par l'objet Application. Vous y trouverez notamment l'événement WindowActivate, correspondant à l'activation d'une fenêtre par l'utilisateur, ou encore WorkbookBeforePrint, qui survient avant l'exécution d'une impression. Plus généralement, les événements dont le nom contient la chaîne Before surviennent avant l'exécution d'une tâche. Ces événements intègrent généralement un argument Cancel permettant d'annuler la tâche. Vous pouvez ainsi créer une procédure

qui s'assurera que les conditions requises par une tâche sont remplies avant d'exécuter cette tâche. Les événements dont le nom contient la chaîne *After* surviennent après l'exécution d'une tâche. Pour des informations supplémentaires sur les événements de l'objet *Application*, choisissez l'événement voulu dans la liste *Procédure*. Sélectionnez ensuite le nom de l'événement dans la fenêtre *Code* et appuyez sur la touche *F1*.

Propriétés de l'objet *Application*

Les propriétés de l'objet *Application* peuvent être lues ou modifiées comme les propriétés de n'importe quel objet. Le Tableau 15.1 présente sommairement les propriétés essentielles de l'objet *Application*. Pour plus d'informations, saisissez le nom de la propriété dans une fenêtre *Code*, sélectionnez-le et tapez sur la touche *F1*. Vous accéderez ainsi à la rubrique d'aide de la propriété.

Tableau 15.1 : Les propriétés essentielles de l'objet *Application*

Propriété	Description
ActiveWorkbook	Renvoie l'objet <i>Workbook</i> correspondant au classeur actif (en lecture seule).
ActiveSheet	Renvoie l'objet <i>Worksheet</i> correspondant à la feuille active du classeur spécifié (en lecture seule).
ActiveCell	Renvoie l'objet <i>Range</i> correspondant à la cellule active de la feuille de classeur spécifiée (en lecture seule).
Caption	Renvoie le nom de l'application, qui apparaît dans la barre de titre. La valeur par défaut est "Microsoft Excel". La Figure 15.2 représente une session Excel dans laquelle nous avons redéfini la valeur de la propriété <i>Caption</i> de l'objet <i>Application</i> (<i>Application.Caption = Chaîne-Titre</i>).
Cursor	Permet de paramétrer l'apparence du curseur au cours des différentes phases d'une macro. Quatre constantes correspondent aux quatre curseurs disponibles. Si vous modifiez l'apparence du curseur au cours d'une macro, n'oubliez pas de redéfinir la propriété <i>Cursor</i> à <i>xlDefault</i> à la fin de l'exécution de la macro. Sinon, le curseur conservera l'apparence qui lui a été affectée.
DisplayAlerts	Permet d'éviter l'affichage de messages Excel au cours de l'exécution d'une macro. Paramétrez cette propriété sur <i>False</i> pour éviter l'affichage de messages Excel. N'omettez pas de redéfinir la propriété <i>DisplayAlerts</i> sur <i>True</i> en fin de macro.
DisplayFormulaBar, DisplayScrollBars et DisplayStatusBar	Déterminent respectivement si la barre de formule, les barres de défilement et la barre d'état sont affichées (<i>True</i>) ou non (<i>False</i>).

Tableau 15.1 : Les propriétés essentielles de l'objet *Application* (suite)

Propriété	Description
EnableCancelKey	Détermine si l'utilisateur peut ou non interrompre une macro en cours d'exécution à l'aide de la combinaison Ctrl+Pause. Par défaut, sa valeur est <code>xlInterrupt</code> et l'utilisateur peut interrompre la macro. Pour interdire l'interruption d'une macro par l'utilisateur, définissez cette propriété sur <code>xlDisabled</code> ou sur <code>xlErrorHandler</code> .
PathSeparator	Renvoie le caractère utilisé comme séparateur dans les chemins – un antislash (\) sous Windows et deux-points (:) sous Macintosh. Cette propriété est utile si vous développez des projets VBA tournant sous Windows et sur Macintosh (en lecture seule).
ScreenUpdating	Détermine si l'affichage écran est mis à jour (<code>True</code>) ou non (<code>False</code>) lors de l'exécution d'une macro. Ne pas mettre à jour l'affichage écran lors de l'exécution d'une macro en améliore considérablement les performances. De plus, les tâches effectuées sont invisibles pour l'utilisateur tant que la macro n'est pas terminée. La propriété <code>ScreenUpdating</code> est automatiquement redéfinie à <code>True</code> lorsque la macro s'achève.
ThisWorkbook	Renvoie le classeur contenant le code de la macro qui s'exécute. Il ne s'agit pas forcément du classeur actif (<code>ActiveWorkbook</code>).
UserName	Détermine le nom d'utilisateur. Il s'agit par défaut du nom apparaissant dans l'onglet Général de la boîte de dialogue Options.

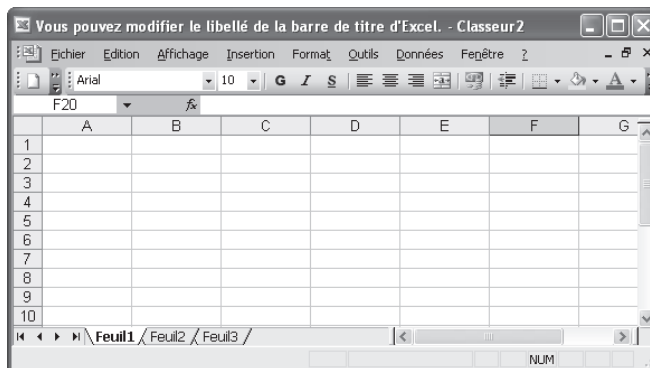


Lorsque vous modifiez les propriétés de l'objet `Application` au cours d'une macro, pensez à les redéfinir à la fin de la macro.

Si vous définissez la propriété `EnableCancelKey` à `False`, vous devez être certain que votre macro n'est pas boguée et gère les exceptions. Si, par exemple, elle exécute une boucle à l'infini, vous n'aurez aucun moyen de l'interrompre.

Figure 15.2

Vous pouvez modifier le titre de la session Excel.



Méthodes de l'objet *Application*

Le Tableau 15.2 présente les méthodes les plus intéressantes de l'objet *Application*.

Tableau 15.2 : Les méthodes essentielles de l'objet *Application*

<i>Méthode</i>	<i>Description</i>
Calculate	Effectue un calcul forcé sur tous les classeurs ouverts. Utilisez cette méthode pour mettre à jour les classeurs ouverts lorsque Excel est paramétré pour effectuer des calculs manuels (onglet Calcul de la boîte de dialogue Options).
OnKey	Exécute une procédure lorsqu'une combinaison de touches est activée. Pour plus d'informations, consultez l'aide en ligne de VBA.
OnTime	Programme l'exécution d'une procédure à un moment précis (ce soir à 20 h 30 par exemple). Pour plus d'informations, consultez l'aide en ligne de VBA.
OnUndo	Définit le texte placé derrière la commande Annuler du menu Édition et la procédure exécutée lorsque l'utilisateur sélectionne cette commande. Pour plus d'informations, consultez l'aide en ligne de VBA.

L'objet *ThisWorkbook*

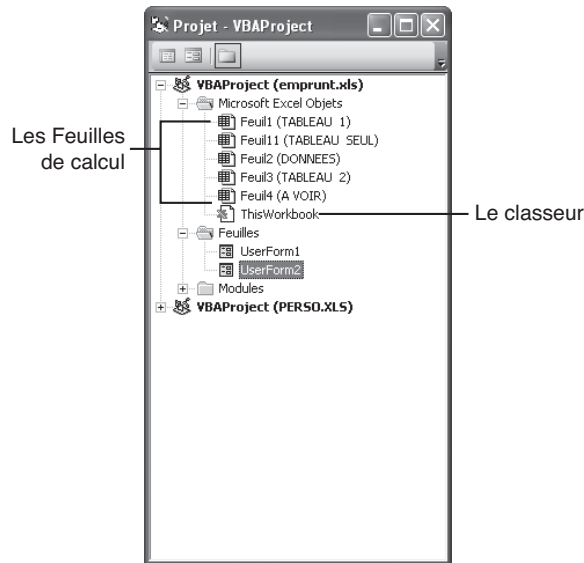
Disponible dans le dossier Microsoft Excel Objets de l'Explorateur de projet, l'objet *ThisWorkbook* représente le classeur correspondant au projet affiché dans l'Explorateur de projet (voir Figure 15.3). Au même titre que l'objet *Application*, il intègre des propriétés et des méthodes permettant de le manipuler et de se renseigner sur son état. Il prend aussi en charge des événements qui permettront d'affecter des procédures événementielles gérant les événements utilisateur susceptibles d'affecter le classeur.

Pour créer une procédure événementielle pour l'objet *ThisWorkbook*, double-cliquez sur *ThisWorkbook* dans l'Explorateur de projet. Dans la zone Objet de la fenêtre Code, sélectionnez *Workbook*. Les instructions d'encadrement d'une procédure événementielle *Open* (ouverture du classeur) sont automatiquement insérées.

L'objet *ThisWorkbook* est un objet *Workbook* (classeur). Il possède donc les mêmes propriétés, et les mêmes méthodes que tous les objets de ce type. Il prend en charge un nombre important d'événements, tels que l'événement *Open* correspondant à l'ouverture du classeur ou l'événement *SheetActivate* correspondant à l'activation d'une autre feuille de calcul du classeur. Certains de ces événements prennent des arguments permettant de connaître précisément l'action effectuée par l'utilisateur. Par exemple, l'événement *SheetActivate* prend pour argument un objet *Worksheet* qui correspond à la feuille activée.

Figure 15.3

*Les objets du dossier
Microsoft Excel Objets
correspondent au
classeur et à ses feuilles
de calcul.*



Pour une liste détaillée des propriétés, méthodes et événements des objets Workbook, saisissez **Workbook** dans une fenêtre Code. Sélectionnez ensuite ce mot et appuyez sur la touche **F1**. Vous accédez à la rubrique d'aide de l'objet Workbook. Les liens **Propriétés**, **Méthodes** et **Événements** affichent une liste des membres correspondants de l'objet Workbook.

Ouvrez n'importe quel classeur Excel. Ouvrez la fenêtre Code de l'objet **ThisWorkbook** du projet de ce classeur et placez-y le code suivant :

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox "La feuille " & Sh.Name & " a été activée.", _
        vbOKOnly + vbInformation, "Événement SheetActivate détecté"
End Sub
```

Retournez dans Excel et activez les différentes feuilles. Une boîte de dialogue indiquant le nom de la feuille sélectionnée s'affiche chaque fois que vous activez une feuille. Cette fonction est intéressante si vous souhaitez limiter l'accès à une feuille de calcul à certains utilisateurs. La procédure suivante affiche une boîte de dialogue dans laquelle l'utilisateur est invité à entrer un mot de passe s'il tente d'accéder à la feuille "Clients".

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    Application.EnableCancelKey = xlDisabled
    If Sh.Name = "Clients" Then
        ActiveWindow.Visible = False
        Dim MotDePasse As String
        MotDePasse = InputBox("Entrez votre mot de passe.", _
```

```

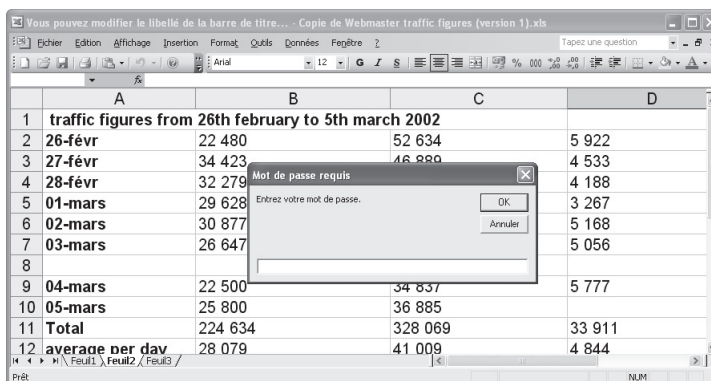
"Mot de passe requis")
If Not MotDePasse = «jk85m» Then
    MsgBox "Le mot de passe saisi est incorrect.", _
        vbOKOnly + vbInformation, "Mot de passe incorrect"
    ThisWorkbook.Sheets(«Feuil2»).Activate
End If
Windows(«Représentants par clients.xlsx»).Visible = True
End If
End Sub

```

Notez que nous avons défini la propriété `EnableCancelKey` de l'objet `Application` à `xlDisabled`, de façon que l'utilisateur ne puisse pas interrompre la macro. La propriété `Visible` de l'objet `ActiveWindow` (la fenêtre active) est définie à `False` de manière à masquer la feuille de classeur tant que l'utilisateur n'a pas entré le mot de passe. Si l'utilisateur saisit un mauvais mot de passe, la feuille `Feuil2` du classeur est activée. Enfin, la propriété `Visible` de la fenêtre correspondant au classeur est définie à `True` de façon à afficher de nouveau le classeur.

Figure 15.4

Vous pouvez protéger l'accès aux données d'une feuille de calcul Excel.



Pour empêcher l'utilisateur de lire le mot de passe dans Visual Basic Editor, protégez aussi votre projet par mot de passe.

Pour des informations supplémentaires sur les événements de l'objet `Application`, choisissez l'événement voulu dans la liste Procédure. Sélectionnez ensuite le nom de l'événement dans la fenêtre Code et appuyez sur la touche F1.

L'objet *Worksheet*

Le dossier Microsoft Excel Objets contient aussi des objets correspondant aux feuilles du classeur (voir Figure 15.3). Ces objets sont des objets *Worksheet* (feuille de calcul) et possèdent donc les mêmes propriétés et méthodes que les objets de ce type. Utilisez ces objets pour créer des procédures événementielles spécifiques aux feuilles.

Les feuilles gèrent les événements présentés dans le Tableau 15.3. Pour accéder aux rubriques d'aide des événements de l'objet *Worksheet*, ouvrez la fenêtre Code d'un objet Feuille. Choisissez l'événement voulu dans la liste Procédure, puis sélectionnez le nom de l'événement dans la fenêtre Code et appuyez sur la touche F1.

Tableau 15.3 : Événements gérés par les objets *Worksheet*

<i>Événement</i>	<i>Description</i>
<code>Activate</code>	Survient lorsque la feuille de calcul est activée.
<code>BeforeDoubleClick</code>	Survient avant le double-clic sur une feuille de calcul ou un graphique. La procédure reçoit l'objet <code>Range</code> correspondant à la cellule sur laquelle l'utilisateur a double-cliqué. Utilisez l'argument <code>Cancel</code> pour annuler le double-clic si nécessaire.
<code>BeforeRightClick</code>	Survient avant que le clic droit ne soit validé. La procédure reçoit l'objet <code>Range</code> correspondant à la cellule sur laquelle l'utilisateur a cliqué du bouton droit. Utilisez l'argument <code>Cancel</code> pour annuler le double-clic si nécessaire.
<code>Calculate</code>	Survient lorsqu'un calcul est effectué. C'est le cas si Excel est paramétré pour un calcul automatique et si la valeur d'une cellule utilisée comme argument dans des fonctions de la feuille est modifiée. Si Excel est paramétré pour un calcul manuel, l'événement survient lorsque l'utilisateur force le calcul.
<code>Change</code>	Survient lorsque la valeur d'une cellule est modifiée. La valeur d'une cellule est considérée comme modifiée après que l'utilisateur a sélectionné une autre cellule. La procédure reçoit l'objet <code>Range</code> correspondant à la cellule dont la valeur a été changée.
<code>Deactivate</code>	Survient lorsque la feuille de calcul est désactivée, c'est-à-dire lorsque l'utilisateur sélectionne une autre feuille de calcul.
<code>SelectionChange</code>	Survient lorsque l'utilisateur modifie la cellule ou la plage de cellules sélectionnée. La procédure reçoit l'objet <code>Range</code> correspondant à la plage sélectionnée.

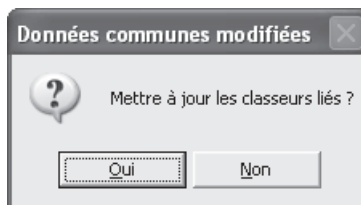
La procédure suivante affiche une boîte de dialogue invitant l'utilisateur à mettre à jour les classeurs liés s'il modifie la valeur de la cellule B5 de la feuille de classeur. S'il clique sur

le bouton Oui, la procédure ProcédureMiseAJour est appelée et la valeur de la cellule B5 lui est passée comme argument.

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
    If Target.Address = "$B$5" Then
        Dim MiseAJour As Integer
        MiseAJour = MsgBox("Mettre à jour les classeurs liés ?", _
            vbYesNo + vbQuestion, "Données communes modifiées")
        If MiseAJour = vbYes Then
            Call ProcédureMiseAJour(Target.Value)
        End If
    End If
End Sub
```

Figure 15.5

S'il modifie la valeur de la cellule B5, l'utilisateur est invité à mettre à jour les classeurs liés.



Notez que les fenêtres Code des objets Workbook et Worksheet ne peuvent contenir que des procédures événementielles. Les procédures appelées doivent se trouver dans des modules de code.

Pour une liste détaillée des propriétés, méthodes et événements des objets Worksheet, saisissez Worksheet dans une fenêtre Code.

Sélectionnez ensuite ce mot et appuyez sur la touche F1. Vous accédez à la rubrique d'aide de l'objet Worksheet. Les liens Propriétés, Méthodes et Événements affichent une liste des membres correspondants de l'objet Sheet.



16

Protéger et authentifier des projets VBA

Au sommaire de ce chapitre

- Les virus macros
- Se protéger des virus macros
- Protéger l'accès aux macros
- Authentifier ses macros

Les virus macro

Les virus macros font partie des virus les plus fréquents. Il s'agit en général de macros conçues pour s'exécuter à l'ouverture d'un document ou pour se substituer à certaines commandes de l'application hôte. Ces virus macros sont attachés à un document et sont susceptibles d'affecter les documents du même type sur une machine infectée.

Du fait de leur grande souplesse de personnalisation et de leur popularité, Word et Excel sont les applications les plus touchées par les virus macros. Les virus macros sont cependant susceptibles de toucher tout type de document capable de stocker des macros.

Les virus macros peuvent être plus ou moins nuisibles. Les virus les plus néfastes peuvent supprimer des fichiers présents sur le disque dur ou sur le réseau, ou endommager des fichiers indispensables au bon fonctionnement d'une application, voire du système. Plus pernicieux, un virus macro peut supprimer des données dans un document ou au contraire ajouter des mots, et en modifier ainsi le sens.

Se protéger des virus macros

Dans la version 97 d'Office, la seule option de protection contre les virus macros consistait à prévenir l'utilisateur de la présence de macros dans un document au moment de son ouverture, et à lui proposer d'activer ou de désactiver les macros contenues dans le document. Cependant, si l'on désactivait les macros, le document était ouvert en lecture seule, et il était alors impossible d'y apporter des modifications.

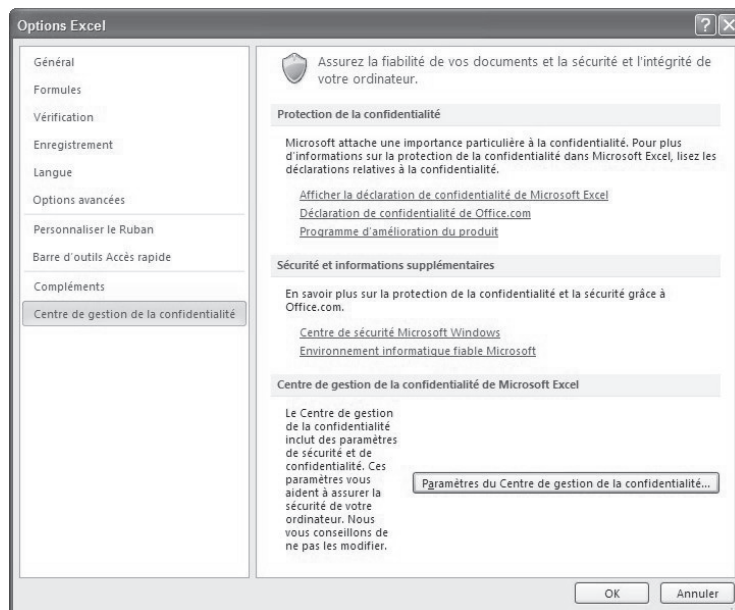
Depuis la version 2000 d'Office, Microsoft a introduit deux nouveautés dédiées à la sécurité et à la protection contre les virus macros : la possibilité de définir des niveaux de sécurité, comparables à ceux qui existaient déjà dans Internet Explorer, et la signature électronique des macros VBA.

Définir un niveau de sécurité dans Excel 2010 et Excel 2007

Les options de sécurité ont sensiblement été modifiées dans les versions 2010 et 2007 d'Excel. Pour définir les options de sécurité, procédez comme suit :

1. Cliquez sur l'onglet Fichier (le bouton Office avec Excel 2007) puis, dans le volet qui s'affiche, sur le bouton Options Excel.
2. Dans le volet gauche de la fenêtre Options Excel, sélectionnez Centre de gestion de la confidentialité. La fenêtre correspondante s'affiche (voir Figure 16.1).

Figure 16.1
Définissez les options de sécurité d'Excel dans cette fenêtre.



3. Cliquez sur le bouton Paramètres de gestion du centre de confidentialité.

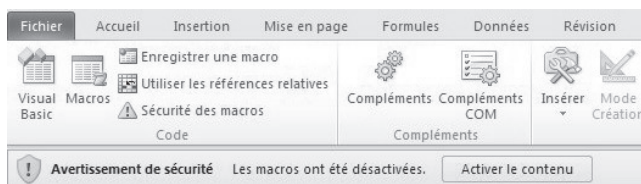
La fenêtre Centre de gestion de la confidentialité est composée d'onglets. Nous présentons sommairement ceux qui nous intéressent ci-après (pour plus d'informations, consultez l'aide d'Excel) :

- **Éditeurs approuvés.** Il s'agit des éditeurs dont vous reconnaissez les signatures numériques dignes de confiance. Les macros des documents signés par ces éditeurs seront activées.
- **Emplacements approuvés.** Vous trouverez ici la liste des dossiers approuvés. Les macros des documents placés dans ces dossiers seront activées. Vous pouvez ajouter des dossiers à la liste par défaut.
- **Documents approuvés.** À l'instar des emplacements approuvés, les macros des documents approuvés sont activées.
- **Compléments.** Vous pouvez définir dans cette fenêtre les options d'activation des compléments Excel – des modules qui ajoutent des fonctions aux programmes Office. Excel est livré avec un certain nombre de compléments, tels que le Solveur ou l'Assistant Somme conditionnelle. Pour connaître les compléments installés, en ajouter ou en supprimer, retournez à la fenêtre Options Excel et activez le volet Compléments.

- **Paramètres ActiveX.** Il s'agit des options d'activation des contrôles ActiveX.
- **Paramètres des macros.** Définissez ici les règles d'activation des macros :
 - Désactiver toutes les macros sans notification : les macros attachées aux documents qui ne se trouvent pas à un emplacement approuvé seront désactivées sans que l'utilisateur en soit averti.
 - Désactiver toutes les macros avec notification (valeur par défaut) : les macros des documents situés hors d'un emplacement approuvé seront désactivées, et l'utilisateur en sera averti par un message affiché dans la barre de message (voir Figure 16.2). Le bouton Activer le contenu (Options dans Excel 2007) permet d'activer les macros du document pour la session en cours.

Figure 16.2

Par défaut, les macros sont désactivées et l'utilisateur en est averti via la barre de messages.



- Désactiver toutes les macros à l'exception des macros signées numériquement : les macros situées à un emplacement autorisé ou signées numériquement par un éditeur que vous avez approuvé (voir ci-dessus) seront automatiquement activées. Dans le cas contraire, l'utilisateur pourra choisir d'activer ou non les macros.
- Activer toutes les macros : les macros seront activées sans aucun contrôle du Centre de confidentialité. Pratique lorsque vous développez des projets VBA et ne souhaitez pas avoir à activer les macros manuellement, cette option est déconseillée avec les documents provenant de sources extérieures.
- **Mode protégé.** Il permet d'ouvrir les fichiers potentiellement dangereux en en désactivant les macros.
- **Barre de message.** Permet de définir si un message s'affiche ou non lorsque le contenu actif d'un document ouvert est désactivé.
- **Contenu externe.** On définit ici le comportement d'Excel lorsque les macros établissent des connexions avec des sources externes, c'est-à-dire avec d'autres fichiers. Il peut s'agir de fichiers Excel ou d'autres types de données.

Astuce

Pour accéder directement aux options de sécurité des macros, cliquez sur le bouton Sécurité des macros de l'onglet Développeur.

Définir un niveau de sécurité avec Excel 2000, XP et 2003

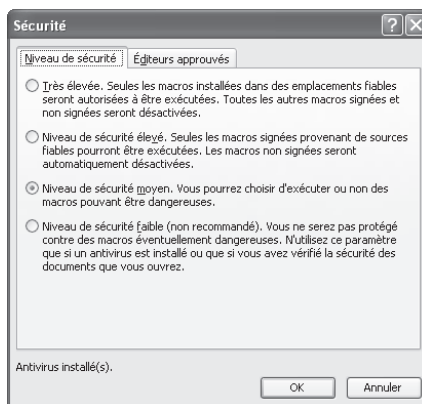
Les niveaux de sécurité permettent de définir les conditions dans lesquelles les macros contenues dans un document seront activées ou désactivées. Vous pouvez ainsi définir des niveaux de sécurité adaptés aux différents utilisateurs, en leur laissant ou non décider de la fiabilité des macros contenues dans un document.

Excel 2003 permet de choisir parmi quatre niveaux de sécurité tandis que les versions antérieures proposent trois options. Pour définir un niveau de sécurité, procédez comme suit :

1. Choisissez Outils > Macro > Sécurité afin d'afficher la boîte de dialogue Sécurité (voir Figure 16.3).

Figure 16.3

Définissez un niveau de sécurité dans la boîte de dialogue Sécurité.



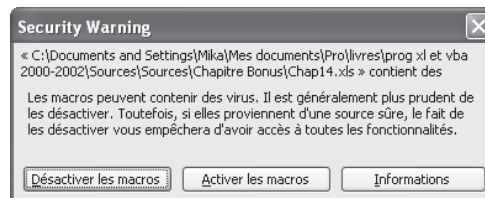
2. Activez l'onglet Niveau de sécurité si nécessaire et cochez l'une des options de sécurité disponibles :
 - **Niveau de sécurité très élevé (Office 2003 seulement).** C'est le lieu de stockage des macros qui en définit la fiabilité. Seules les macros de documents stockés dans des emplacements définis comme fiables seront exécutées.
 - **Niveau de sécurité élevé.** Seules les macros signées numériquement, c'est-à-dire authentifiées par leur développeur, et dont la signature a été ajoutée à la liste des sites dignes de confiance seront activées. L'utilisateur choisira d'activer ou de désactiver les macros signées mais dont la signature ne fait pas partie de la liste des sources fiables. Toutes les autres macros non signées seront désactivées sans que l'utilisateur en soit prévenu.
 - **Niveau de sécurité moyen.** Les macros signées seront traitées de la même façon que dans le cas d'un niveau de sécurité haut. Par contre, l'utilisateur sera averti de la présence de macros non signées et invité à activer ou à désactiver ces macros (voir Figure 16.4).

- **Niveau de sécurité bas.** Toutes les macros seront activées, qu'elles soient signées ou non.

3. Validez en cliquant sur OK.

Figure 16.4

Le niveau de sécurité moyen laisse à l'utilisateur le soin d'activer ou de désactiver les macros non identifiées comme provenant d'une source fiable.



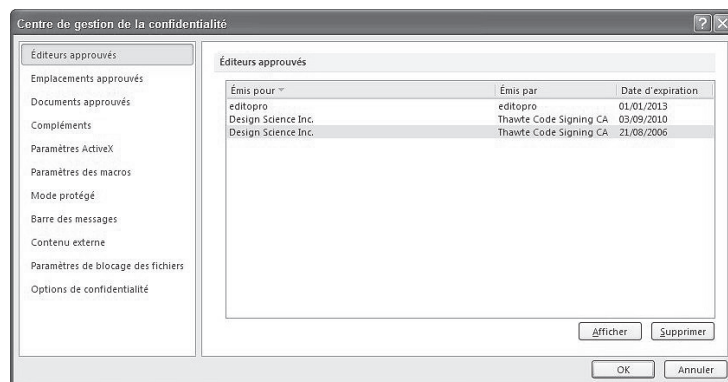
Lorsque des macros sont désactivées (de façon automatique dans le cas d'un niveau de sécurité haut, ou par choix de l'utilisateur dans le cas d'un niveau de sécurité moyen), celles-ci restent tout de même accessibles dans la boîte de dialogue Macros. Si vous tentez d'exécuter une macro désactivée, un message vous informe que vous ne pouvez pas exécuter les macros stockées dans ce document. Pour pouvoir exécuter les macros, vous devez alors fermer le document, redéfinir un niveau de sécurité moyen ou bas si nécessaire, puis rouvrir le document.

Les signatures numériques

Les signatures numériques permettent à leurs auteurs de certifier la provenance des macros qu'ils distribuent. Pour ajouter une signature numérique à la liste des sources fiables, sélectionnez l'onglet Éditeurs approuvés du Centre de gestion de la confidentialité. Les macros portant les signatures listées ici sont considérées comme fiables. Elles sont donc activées sans que vous soyez prévenu de leur existence, et ce quel que soit le niveau de sécurité choisi (Figure 16.5).

Figure 16.5

Les signatures numériques considérées comme sources fiables apparaissent ici.





Pour consulter la liste des sources fiables avec une version d'Excel antérieure à 2007, choisissez Outils > Macro > Sécurité, puis activez l'onglet Sources fiables de la boîte de dialogue Sécurité.



Choisissez un niveau de sécurité adapté à vos utilisateurs et à votre utilisation des macros. Si vous travaillez dans une entreprise qui développe ses propres macros et n'utilise a priori pas de macros issues de sources extérieures, la signature des macros de la société associée à un niveau de sécurité haut constituera un choix adapté. Le niveau de sécurité élevé constitue également le bon choix pour des utilisateurs qui ne sont pas conscients des dangers liés aux virus macros.

Par contre, si vous utilisez des macros provenant de sources extérieures variables, le niveau de sécurité élevé entraînera la désactivation de toutes les macros non signées, et ce sans que l'utilisateur ne soit prévenu de l'existence de ces macros. Un niveau de sécurité moyen se révélera alors plus adapté, à condition de sensibiliser les utilisateurs aux dangers des virus macros et de fixer des règles pour l'activation ou la désactivation des macros.

Le niveau de sécurité bas ne devrait être appliqué que dans des conditions exceptionnelles. Par exemple, si vous développez un nombre important de macros, que vous ne disposiez pas d'une signature numérique, et que vous souhaitiez que les macros des documents que vous ouvrez soient toujours activées, vous pourrez activer le niveau de sécurité le plus bas. Il est alors recommandé de travailler sur une machine communiquant le moins possible avec l'extérieur, équipée d'un antivirus récent, et d'effectuer des sauvegardes quotidiennes de vos données.

Si la définition d'un niveau de sécurité adapté constitue une barrière de sécurité importante, il est fortement recommandé de compléter cette protection par l'utilisation d'un antivirus. Depuis la version 2000, Office intègre une nouvelle technologie qui permet aux antivirus de fonctionner depuis Office. Les antivirus compatibles Office peuvent ainsi vérifier si un document contient des virus avant son ouverture.

Effectuer des sauvegardes des macros

Vous pouvez prendre toutes les précautions imaginables, définir le niveau de sécurité le plus élevé et installer le tout dernier antivirus compatible Office, vous ne serez jamais à l'abri d'un virus qui contourne les barrières mises en place, ou d'un simple "crash" de votre disque dur ! Si vous avez passé des jours, voire des mois à développer des programmes VBA et si vous vous retrouvez du jour au lendemain sans la moindre ligne de code, vous risquez fortement de détruire votre ordinateur.

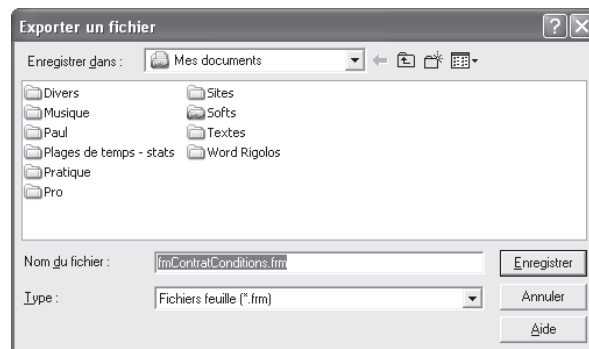
Ne prenez pas de risques inutiles et respectez cette règle incontournable : effectuez des sauvegardes régulières de vos données sensibles. En sauvegardant les classeurs et les modèles auxquels sont attachées vos macros, vous sauvegarderez également le projet VBA correspondant. Le classeur PERSONAL.XLSB contiendra probablement l'essentiel de vos macros Excel. Une autre solution consiste à exporter les modules et les feuilles et à sauvegarder les fichiers ainsi générés en lieu sûr.

Pour exporter un module ou une feuille :

1. Sélectionnez le module ou la feuille voulu dans l'Explorateur de projet.
2. Cliquez du bouton droit et, dans le menu contextuel qui s'affiche, sélectionnez Exporter un fichier.
3. La boîte de dialogue Exporter un fichier s'affiche (voir Figure 16.6). Le type du fichier exporté varie selon l'élément supprimé :
 - Les modules standard ou modules de code sont exportés sous la forme de fichiers Basic portant l'extension .bas. Les fichiers .bas sont des fichiers texte, et peuvent donc être consultés et modifiés dans un éditeur de texte tel que le Bloc-notes de Windows.
 - Les feuilles sont exportées sous la forme de deux fichiers Feuille portant les extensions .frm et .frx. Le premier est un fichier texte contenant quelques données propres à la feuille (telles que son nom, sa taille, sa position lors de l'affichage), ainsi que le code qui lui est attaché. Le second ne peut être lu dans un éditeur de texte et contient toutes les autres informations (essentiellement les données propres aux différents contrôles de la feuille).
 - Les modules de classe sont exportés sous la forme de fichiers Classe portant l'extension .cls.
4. Indiquez le répertoire et le nom d'enregistrement, puis cliquez sur le bouton Enregistrer.

Figure 16.6

Indiquez le nom du fichier exporté et son dossier d'enregistrement.



Une fois un fichier exporté, vous pouvez l'importer dans n'importe quel projet. Pour importer les modules et les feuilles dans un projet, affichez la fenêtre de l'Explorateur de projet, cliquez droit sur l'un des éléments du projet et choisissez la commande Importer un fichier. Sélectionnez ensuite le fichier à importer (.bas, .frm ou .cls).

Protéger l'accès aux macros

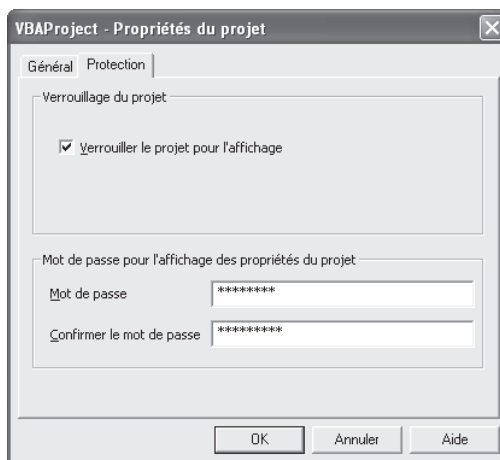
Il est parfois nécessaire de protéger ou de limiter l'accès aux macros. La protection des macros peut consister à interdire l'accès au code d'un projet pour en préserver l'intégrité, ou à limiter le droit d'exécution des macros.

Verrouiller un projet

Si d'autres utilisateurs accèdent à vos macros, ou si vous installez des macros sur d'autres postes et que vous ne souhaitez pas que le code puisse être modifié, vous pouvez simplement protéger l'accès au code du projet par mot de passe. Pour verrouiller un projet, procédez comme suit :

1. Affichez l'Explorateur de projet de façon à y visualiser le projet.
2. Cliquez droit sur le nom du projet et sélectionnez la commande Propriétés de *projet*.
3. Activez l'onglet Protection et cochez la case Verrouiller le projet pour l'affichage (voir Figure 16.7).
4. Dans les zones définies à cet effet, saisissez à deux reprises le mot de passe qui sera ensuite demandé pour accéder au projet, puis validez.

Figure 16.7
*Protégez votre projet
par un mot de passe.*

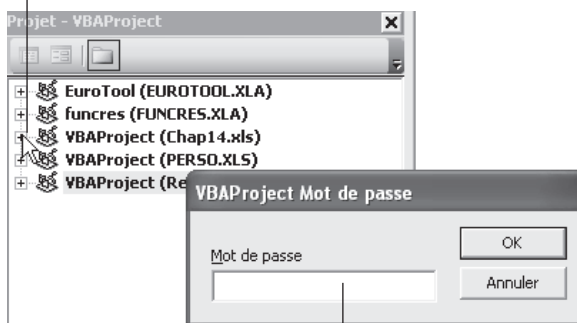


Le verrouillage du projet prendra effet dès la prochaine ouverture du document. Lorsqu'un projet est verrouillé par un mot de passe, ce mot de passe est nécessaire pour accéder au code du projet ou pour supprimer une macro (voir Figure 16.8).

Figure 16.8

Le mot de passe est maintenant requis pour accéder au code du projet.

Lorsque l'utilisateur tente d'accéder au projet...



... le mot de passe lui est demandé



Si vous oubliez le mot de passe défini au moment du verrouillage du projet, vous n'aurez aucun moyen d'accéder au code de ce projet. Il est fortement recommandé d'effectuer une sauvegarde du projet ou d'en exporter les modules et les feuilles avant de le protéger par mot de passe.

Limiter les droits d'exécution d'une macro

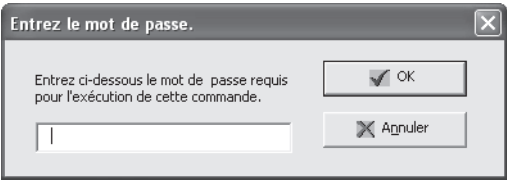
Il est parfois nécessaire de définir des autorisations pour l'exécution d'une macro. Une macro Excel peut par exemple permettre d'accéder à des informations financières confidentielles. Une macro peut également causer des dégâts si elle n'est pas utilisée correctement ; son utilisation doit alors être limitée aux utilisateurs initiés.

La solution consiste alors à réclamer un mot de passe au moment de l'exécution de la macro. On crée pour cela une feuille intégrant une zone de texte dans laquelle l'utilisateur est invité à entrer le mot de passe. Une procédure vérifie que le mot de passe fourni est correct avant d'exécuter le code. Le projet devra évidemment être protégé contre l'affichage, afin que l'utilisateur ne puisse pas consulter le mot de passe dans le code de la feuille.

Voici une façon d'écrire le code de vérification du mot de passe. Créez une feuille UserForm et placez sur celle-ci un contrôle Label1, un contrôle TextBox et deux contrôles CommandButton (votre feuille doit ressembler à celle qui est présentée à la Figure 16.9). Définissez-en les propriétés comme indiqué ci-après.

Propriété	Valeur
Feuille	
Name	fmMotDePasse
Caption	Entrez le mot de passe
Contrôle Label1	
Name	LbMotdePasse
Caption	Entrez ci-après le mot de passe requis pour l'exécution de cette commande
Contrôle TextBox	
Name	TxtMotDePasse
PasswordChar	*
Contrôle CommandButton 1	
Name	CmdOK
Caption	OK
Default	True
Contrôle CommandButton 2	
Name	CmdAnnuler
Caption	Annuler
Cancel	True

Figure 16.9
La feuille fmMotDePasse en mode Exécution.



Insérez ensuite le code suivant dans la fenêtre Code de la feuille.

```
1: Private Sub UserForm_Initialize()  
2:     txtMotDePasse.Value = ""  
3:     txtMotDePasse.SetFocus  
4: End Sub  
5:  
6: Private Sub cmdAnnuler_Click()  
7:     Unload Me
```

```
8:      MsgBox "Cette commande ne peut être exécutée sans le mot de passe.", _
9:      vbOKOnly + vbExclamation, "Fin de la commande"
10:     End
11: End Sub
12:
13: Private Sub cmdOK_Click()
14:     'On vérifie si le mot de passe entré est bon.
15:     If txtMotDePasse.Text = "ftg87" Then
16:         Unload Me
17:     Else
18:         MsgBox «Le mot de passe fourni n'est pas correct.», _
19:         vbOKOnly + vbExclamation, «Mot de passe incorrect»
20:         txtMotDePasse.Value = «»
21:         txtMotDePasse.SetFocus
22:     End If
23: End Sub
```

Pour tester ce programme, insérez la procédure suivante dans n'importe quel module de code et exécutez-la.

```
Sub TestMotDePasse()
    fmMotDePasse.Show
    MsgBox "Si ce message s'affiche, c'est que le programme se poursuit."
End Sub
```

Cette procédure appelle la feuille de vérification du mot de passe (fmMotDePasse). Elle ne reprendra la main si le mot de passe entré dans la feuille est correct, et affichera alors un message indiquant que le code se poursuit.

La première procédure (lignes 1 à 4) s'exécute à chaque affichage de la feuille. Elle affecte une chaîne vide à la zone de texte et lui passe le focus.

La procédure suivante (lignes 6 à 11) s'exécute lorsque l'utilisateur clique sur le bouton libellé Annuler ou tape sur la touche Échap. La feuille est alors déchargée et un message informe l'utilisateur que la commande prend fin. L'instruction End met fin à l'exécution du programme.

La procédure cmdOK_Click se déclenche lorsque l'utilisateur clique sur le bouton OK ou tape sur la touche Entrée (à condition que le focus ne soit pas sur le bouton Annuler). Lignes 15 à 22, une structure conditionnelle If...Then...Else vérifie que le mot de passe entré est correct, c'est-à-dire que l'utilisateur a saisi ftg87 (ligne 15) dans la zone de texte. Si tel est le cas, la feuille est déchargée et la procédure ayant appelé l'affichage de la feuille reprend la main. Dans le cas contraire, un message indique à l'utilisateur que le mot de passe entré est incorrect. La zone de texte est alors vidée de la valeur précédemment entrée et reprend le focus.



Lignes 7 et 16, nous avons utilisé le mot clé Unload et non Hide pour faire disparaître la feuille. Rappelez-vous que Hide masque la feuille mais ne libère pas les ressources mémoire qu'elle exploite. Si vous remplacez l'instruction Unload Me de la ligne 16 par l'instruction Me.Hide, la prochaine fois que la feuille sera affichée au cours de la session, le mot de passe sera déjà affiché dans la zone de texte. Il suffira alors à l'utilisateur de cliquer sur le bouton OK pour exécuter la macro.

Le programme présente ici un bogue grave : si l'utilisateur clique sur la croix de fermeture de la fenêtre, aucun événement n'est détecté. La feuille se ferme et le programme se poursuit, comme si le mot de passe avait été fourni. Pour remédier à ce problème, ajoutez au code de la feuille la procédure suivante :

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
        Me.Hide
        MsgBox "Cette commande ne peut être exécutée sans le mot de passe.", _
            vbOKOnly + vbExclamation, "Fin de la commande"
    End
End If
End Sub
```

L'événement QueryClose est détecté lorsqu'une tentative de fermeture de la fenêtre se produit. L'argument Cancel permet d'annuler la fermeture. L'argument CloseMode indique la cause de l'événement. Ici, la procédure est exécutée si l'événement correspond à un clic sur le bouton de fermeture de la fenêtre (CloseMode = vbFormControlMenu). Si vous ne précisez pas l'argument CloseMode pour lequel vous souhaitez que la procédure s'exécute, elle s'exécutera dans n'importe quel contexte, y compris lorsque la méthode Unload sera utilisée dans le code. Dans ce cas, si l'utilisateur clique sur le bouton Annuler, le message l'informant que le programme prend fin sera affiché à deux reprises – dans le cadre des procédures événementielles cmdAnnuler_Click et UserForm_QueryClose.

Dans ce premier exemple, la feuille reste affichée jusqu'à ce que l'utilisateur entre le mot de passe correct ou qu'il clique sur le bouton Annuler. Vous pouvez également choisir de limiter le nombre d'essais de saisie du mot de passe. La procédure cmdOK_Click reproduite ci-après a été modifiée de sorte que le programme prenne fin après trois tentatives infructueuses de saisie du mot de passe (les modifications apparaissent en gras dans le code). Nous avons également redéfini la propriété Caption de la feuille à "Entrez le mot de passe. Tentative 1 sur 3" et modifié le code de manière que cette information soit mise à jour dans la barre de titre de la fenêtre à chaque nouvelle tentative.

```
1: Private Sub cmdOK_Click()
2: 'La variable compteur servira à compter le nbre de tentatives.
```

```

3: Static compteur As Byte
4: compteur = compteur + 1
5:
6:   If txtMotDePasse.Text = "ftg87" Then
7:       Unload Me
8:   Else
9:
10:      'Si c'est la 3e fois que l'utilisateur entre un mot
11:      'de passe incorrect, le programme prend fin
12:      If compteur = 3 Then
13:          MsgBox «Echec dans la saisiedu mot de passe.» & _
14:              vbCr & «La commande ne peut être exécutée», _
15:              vbOKOnly + vbExclamation, «Mot de passe incorrect»
16:          End
17:      End If
18:
19:      MsgBox «Le mot de passe fourni n'est pas correct.», _
20:          vbOKOnly + vbExclamation, «Mot de passe incorrect»
21:      txtMotDePasse.Value = «»
22:      txtMotDePasse.SetFocus
23:      Me.Caption = «Entrez le mot de passe. Tentative « & _
24:                  compteur + 1 & « sur 3»
25:
26:
27:   End If
28:
29: End Sub

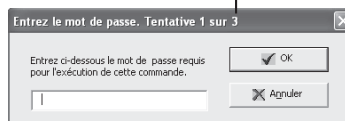
```

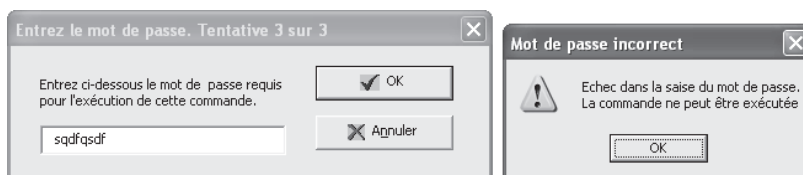
Exécutez la feuille. Lorsque vous saisissez un mot de passe incorrect, un message vous en informe et le titre de la fenêtre est mis à jour de façon à refléter le nombre de tentatives (voir Figure 16.10). Si l'utilisateur saisit trois fois de suite un mot de passe erroné, un message l'informe que le programme ne peut être exécuté et le programme prend fin (voir Figure 16.11).

Figure 16.10

Le nombre de tentatives s'affiche dans la barre de titre de la fenêtre.

La barre de titre indique le nombre de tentatives effectuées



**Figure 16.11**

Après trois tentatives infructueuses, le programme prend fin.

La variable `Compteur` est déclarée à l'aide du mot clé `Static` en début de procédure. Notre variable est ainsi incrémentée de 1 à chaque exécution de la procédure, c'est-à-dire chaque fois que l'utilisateur valide un mot de passe. Notez que nous n'avons pas intégré d'instruction affectant la valeur 0 à la variable `Compteur` ; une telle instruction aurait en effet réinitialisé la variable à 0 à chaque exécution de la procédure qui n'aurait alors jamais pris fin.



Une variable statique est une variable qui conserve sa valeur entre les différents appels d'une procédure. Une variable statique au sein d'un module de feuille conserve sa valeur tant que la feuille s'affiche.

Lignes 11 à 17, une structure `If...Then...Else` a été imbriquée à la structure conditionnelle initiale. Elle vérifie si `compteur` est égal à 3, c'est-à-dire si c'est la troisième fois que l'utilisateur propose un mot de passe erroné. Si tel est le cas, un message s'affiche et l'instruction `End` de la ligne 16 interrompt le programme.

Enfin, lignes 23 et 24, nous avons ajouté une instruction qui met à jour le texte affiché dans la barre de titre de la fenêtre. Pour définir quel sera le numéro de la prochaine tentative, on incrémente la valeur de `Compteur` (le nombre de tentatives effectuées, y compris celle en cours) de 1.

Telle qu'elle se présente dans les deux versions du programme que nous avons données ici, la feuille `fmMotDePasse` permet d'exiger de l'utilisateur un mot de passe à tout moment de l'exécution d'un programme. Il suffit pour cela de placer l'instruction :

```
fmMotDePasse.Show
```

à l'endroit où l'on souhaite que le mot de passe soit demandé.

Ce programme peut cependant être rendu plus souple et plus performant en contournant les limitations à un mot de passe (ici `ftg87`) et à trois tentatives, toutes deux définies dans le code de la feuille. Vous souhaitez en effet probablement définir des mots de passe différents pour différents programmes, dont l'utilisation sera limitée à différents utilisateurs.

La solution consiste à appeler une procédure de validation en lui passant les valeurs "mot de passe" et "nombre de tentatives" comme arguments. Celle-ci stockera ces valeurs dans

des variables publiques de niveau module (donc accessibles à l'ensemble des modules du projet) auxquelles les procédures de la feuille pourront accéder.

Créez un nouveau module de code et affectez-lui un nom représentatif, par exemple à MotDePasse. Placez dans la fenêtre de code de celui-ci, le code suivant :

```
1: Public varMotDePasse As String
2: Public varNumTentatives As Byte
3:
4: Sub ControleMotDePasse(MotDePasse As String, NumTentatives As Byte)
5:     varMotDePasse = MotDePasse
6:     varNumTentatives = NumTentatives
7:     fmMotDePasse.Show
8: End Sub
```

Lignes 1 et 2 les variables de niveau module varMotDePasse et varNumTentatives sont déclarées publiques. La procédure ControleMotDePasse (lignes 4 à 8) pourra être appelée par n'importe quelle procédure en lui passant les arguments requis (le mot de passe et le nombre de tentatives), afin de subordonner la poursuite de l'exécution du programme en cours à la présentation de ce mot de passe par l'utilisateur. Cette procédure affecte les valeurs qu'elle reçoit comme arguments aux variables varMotDePasse et varNumTentatives, et appelle ensuite la feuille fmMotDePasse.

Modifiez ensuite le code de la feuille de la façon suivante (les modifications apparaissent en gras) :

```
1: Private Sub UserForm_Initialize()
2:     txtMotDePasse.Value = ""
3:     txtMotDePasse.SetFocus
4:     Me.Caption = «Entrez le mot de passe. Tentative 1» & _
5:     « sur « & varNumTentatives
6: End Sub
7:
8: Private Sub cmdAnnuler_Click()
9:     Me.Hide
10:    MsgBox «Cette commande ne peut être exécutée sans le mot de passe.», _
11:    vbOKOnly + vbExclamation, «Fin de la commande»
12:    End
13: End Sub
14:
15: Private Sub cmdOK_Click()
16:    'La variable compteur servira à compter le nbre de tentatives.
17:    Static compteur As Byte
18:    compteur = compteur + 1
19:
```

```
20: If txtMotDePasse.Text = varMotDePasse Then
21:     Unload Me
22: Else
23:
24:     'Si c'est la Ne fois que l'utilisateur entre un mot
25:     'de passe incorrect, le programme prend fin
26:     If compteur = varNumTentatives Then
27:         MsgBox «Echec dans la saisie du mot de passe.» & _
28:             vbCr & «La commande ne peut être exécutée», _
29:             vbOKOnly + vbExclamation, «Mot de passe incorrect»
30:         End
31:     End If
32:
33:     MsgBox «Le mot de passe fourni n'est pas correct.», _
34:         vbOKOnly + vbExclamation, «Mot de passe incorrect»
35:     txtMotDePasse.Value = «»
36:     txtMotDePasse.SetFocus
37:     Me.Caption = «Entrez le mot de passe. Tentative « & _
38:                 compteur + 1 & « sur « & varNumTentatives
39:
40:
41: End If
42:
43: End Sub
44:
45: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
46:     If CloseMode = vbFormControlMenu Then
47:         Me.Hide
48:         MsgBox «Cette commande ne peut être exécutée sans le mot de passe.», _
49:             vbOKOnly + vbExclamation, «Fin de la commande»
50:     End
51: End If
52: End Sub
```

Partout dans le code où il était fait référence au nombre de tentatives ou au mot de passe, nous avons remplacé les valeurs fixes par des références aux variables `varMotDePasse` et `varNumTentatives`. Nous avons ajouté une instruction définissant le titre de la fenêtre dans la procédure `Initialize` de la feuille (lignes 4 et 5), afin que le texte affiché dans la barre de titre reflète le nombre de tentatives autorisées dès l’affichage de la feuille. Ce titre est ensuite mis à jour à chaque passage de la fenêtre (ligne 38).

Ligne 20 la valeur de la zone de texte est maintenant comparée à la valeur stockée dans la variable `varMotDePasse`, tandis que dans l’instruction de la ligne 26, la variable `Compteur` est maintenant comparée à la valeur de `varNumTentatives`, afin de définir si toutes les tentatives autorisées ont échoué.

Pour appeler le programme mot de passe, il suffit maintenant de placer l'instruction suivante à l'endroit voulu :

```
Call ControleMotDePasse(MotDePasse, NumTentatives)
```

où *MotDePasse* est le mot de passe attendu, et *NumTentatives* le nombre de tentatives autorisées. Pour tester ce programme, placez la procédure suivante dans n'importe quel module du projet et exécutez-la (voir Figure 16.12).

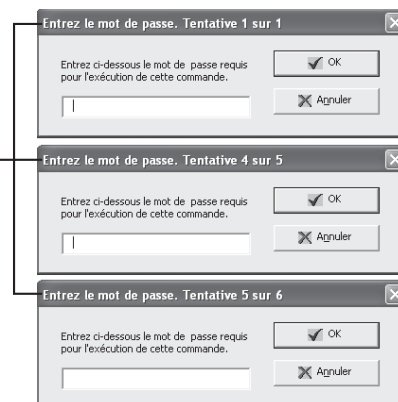
```
1: Sub TestControleMotDePasse()  
2: Dim maVar As String  
3:   Call ControleMotDePasse("Bonjour", 1)  
4:   maVar = Application.UserName  
5:   MsgBox "Mot de passe correct." & vbCrLf & "Poursuite de l'exécution.", _  
6:   vbOKOnly + vbInformation  
7:   Call ControleMotDePasse(maVar, 5)  
8:   maVar = Application.UserInitials & "-vba"  
9:   MsgBox "Mot de passe correct." & vbCrLf & "Poursuite de l'exécution.", _  
10:  vbOKOnly + vbInformation  
11:  Call ControleMotDePasse(maVar, 6)  
12:  MsgBox «Mot de passe correct.» & vbCrLf & «Poursuite de l'exécution.», _  
13:  vbOKOnly + vbInformation  
14: End Sub
```

L'instruction de la ligne 3 appelle le programme de contrôle et attend que le mot de passe Bonjour soit fourni dès la première tentative. L'instruction de la ligne 7 attend que le nom d'utilisateur enregistré pour l'application soit fourni et autorise 5 tentatives. Enfin, ligne 11, le mot de passe passé correspond aux initiales enregistrées pour l'utilisateur en cours, suivies de -vba, et autorise six tentatives. Chaque fois que le bon mot de passe est fourni par l'utilisateur, la procédure *TestControleMotDePasse* reprend la main et un message s'affiche (lignes 5, 9 et 12).

Figure 16.12

Le titre de la fenêtre reflète le nombre d'essais autorisés.

Le nombre de tentatives autorisé est défini lors de l'appel de la procédure de contrôle



Astuce

Vous pouvez également limiter le nombre de caractères autorisés dans la zone de texte de la feuille `fmMotDePasse` au nombre de caractères composant le mot de passe. Insérez pour cela l'instruction suivante dans la procédure `UserForm_Initialize` de la feuille `fmMotDePasse` :

```
txtMotDePasse.MaxLength = Len(varMotDePasse).
```

Rappel

Pour un rappel sur la propriété `MaxLength`, reportez-vous au Chapitre 14.

Une ultime amélioration peut être amenée au programme : autoriser plusieurs mots de passe pour exécuter une macro. Cela peut se révéler nécessaire si vous définissez un mot de passe par "groupe d'utilisateurs". Vous pouvez par exemple limiter l'accès à certaines macros aux utilisateurs du groupe "Financier", et d'autres au groupe "Production". Si le nombre de macros limitées par mots de passe est important, définir un mot de passe par macro deviendra rapidement ingérable, pour vous comme pour les utilisateurs finaux. Il est alors conseillé de définir un mot de passe par groupe. Si une macro est accessible aux deux groupes, le mot de passe de chacun des groupes devra alors être reconnu comme valide.

Voici les modifications à apporter au programme pour autoriser jusqu'à trois mots de passe. Modifiez le code du module `MotDePasse` de la façon suivante :

```
1: Public varMotDePasse1 As String, varMotDePasse2 As Variant, varMotDePasse3
   As Variant
2: Public varNumTentatives As Byte
3:
4: Sub ControleMotDePasse(NumTentatives As Byte, MotDePasse1 As String, Optional
   MotDePasse2 As String, Optional MotDePasse3 As String)
5:     varNumTentatives = NumTentatives
6:     varMotDePasse1 = MotDePasse1
7:
8:     'On vérifie si plusieurs mots de passe ont été passés,
9:     'et on affecte une valeur aux variables en conséquence.
10:    If IsMissing(MotDePasse2) = True Then
11:        varMotDePasse2 = MotDePasse1
12:    Else
13:        varMotDePasse2 = MotDePasse2
14:    End If
15:    If IsMissing(MotDePasse3) = True Then
```

```

16:         varMotDePasse3 = MotDePasse1
17:     Else
18:         varMotDePasse3 = MotDePasse3
19:     End If
20:     fmMotDePasse.Show
21: End Sub

```

Ligne 1, nous avons créé trois variables destinées à recevoir les 3 mots de passe autorisés. Nous avons ensuite modifié la déclaration de la procédure `ControleMotDePasse` (ligne 4) en y ajoutant les arguments optionnels `MotDePasse2` et `MotDePasse3` (déclarés avec le mot clé `Optional`). Notez que nous avons dû déplacer l'argument `Numtentatives` en début de liste car un argument optionnel ne peut être suivi d'un argument obligatoire.

Deux structures conditionnelles (lignes 10 à 14 et lignes 15 à 19) vérifient ensuite si les mots de passe optionnels ont été passés. Si l'argument correspondant au deuxième mot de passe valide a été passé, la variable `varMotDePasse2` se voit affecter la valeur de cet argument. Si l'argument n'a pas été passé, la variable `varMotDePasse2` reçoit la valeur de l'argument `MotDePasse1` qui, lui, est obligatoire. Le même traitement est appliqué à la variable `varMotDePasse3`, fonction du passage ou non de l'argument `MotDePasse3`.



Le mot clé `Optional` indique qu'un argument n'est pas obligatoire. La fonction `IsMissing` peut être utilisée pour vérifier si l'argument a été passé ou non.



La fonction `IsMissing` ne fonctionne correctement qu'avec des arguments de type `Variant`. Si vous déclarez les arguments `MotDePasse2` et `MotDePasse3` de type `String`, la fonction `IsMissing` renverra toujours `False`, que les arguments soient passés ou non. En conséquence, les variables `varMotDePasse2` et `varMotDePasse3` se verront affecter une chaîne vide, qui sera reconnue comme mot de passe valide.

Modifiez ensuite l'instruction :

```
If txtMotDePasse.Text = varMotDePasse Then
```

de la procédure `cmdOK_Click` de façon que chacun des trois mots de passe autorisés soit reconnu comme valide :

```
If txtMotDePasse.Text = varMotDePasse1 Or txtMotDePasse.Text = varMotDePasse2 Or
txtMotDePasse.Text = varMotDePasse3 Then
```

Info

Les instructions des lignes 10 à 19 permettent que les variables varMotDePasse2 et varMotDePasse3 ne soient jamais vides, et ce même si les arguments MotDePasse2 et/ou MotDePasse3 n'ont pas été passés à la procédure ControleMotDePasse. Ce traitement est nécessaire pour qu'une erreur ne soit pas générée lors de la vérification de la comparaison du texte saisi par l'utilisateur dans la feuille aux valeurs contenues dans les variables varMotDePasse2 et varMotDePasse3.

Pour appeler le programme mot de passe, il suffit maintenant de placer l'instruction suivante à l'endroit voulu :

```
Call ControleMotDePasse(NumTentatives, MotDePasse1, MotDePasse2, MotDePasse3)
```

où *NumTentatives* est le nombre de tentatives autorisées et *MotDePasse1*, *MotDePasse2* et *MotDePasse3* sont les mots de passe valides. *MotDePasse2* et *MotDePasse3* sont optionnels. Pour tester ce programme, placez la procédure suivante dans n'importe quel module du projet et exécutez-la.

```
1: Sub TestControlePlusieursMotsDePasse()  
2:   Call ControleMotDePasse(5, «Bonjour»)  
3:   MsgBox «Mot de passe correct.» & vbCr & «Poursuite de l'exécution.», _  
4:   vbOKOnly + vbInformation  
5:   Call ControleMotDePasse(5, «Bonjour», «Salut»)  
6:   MsgBox «Mot de passe correct.» & vbCr & «Poursuite de l'exécution.», _  
7:   vbOKOnly + vbInformation  
8:   Call ControleMotDePasse(5, «Bonjour», «Salut», «Coucou»)  
9:   MsgBox «Mot de passe correct.» & vbCr & «Poursuite de l'exécution.», _  
10:  vbOKOnly + vbInformation  
11: End Sub
```

Chacune des instructions appelant la procédure de vérification du mot de passe (lignes 2, 5 et 8) autorise 5 tentatives. Lors du premier appel, seul le mot de passe Bonjour est passé comme argument. Le deuxième appel passe également la valeur Bonjour à la procédure, mais fournit aussi la valeur Salut pour l'argument optionnel MotDePasse2. L'un ou l'autre des mots de passe sera donc accepté. Enfin, lors du troisième appel, tous les arguments sont passés à la procédure ControleMotDePasse, et les mots de passe Bonjour, Salut et Coucou seront tous trois acceptés. Chaque fois que le bon mot de passe est fourni par l'utilisateur, la procédure TestControlePlusieursMotsDePasse reprend la main et un message s'affiche (lignes 3 et 9).

Authentifier ses macros

Dans les versions antérieures d'Office, le développement des virus macros associé à l'impossibilité d'authentifier l'origine des macros posait un vrai problème pour les utilisateurs amenés à développer des macros nécessaires à l'utilisation de documents qu'ils distribuaient. En effet, lorsqu'un utilisateur ouvrait un document qui lui avait été communiqué et voyait s'afficher un message lui conseillant de désactiver les "dangereuses" macros contenues dans ce document, il choisissait avec raison de suivre les recommandations qui lui étaient données.

La réponse de Microsoft à ce problème a été l'intégration de *signatures numériques*, et ce depuis la version 2000 d'Office. Les signatures numériques identifient la source d'une macro (le nom de la personne ou de la société ayant développé les macros). Lorsque vous ouvrez un document contenant des macros signées numériquement, le détail de la signature de ces macros s'affiche à l'écran.



Le fait que des macros soient numériquement signées ne vous garantit pas de leur fiabilité. Un développeur malhonnête peut se procurer une signature électronique pour signer des virus macros. Ne vous fiez aux signatures électroniques que si elles authentifient une personne ou une société de votre connaissance.

Obtenir une authentification

Afin de pouvoir authentifier vos macros, vous devez vous procurer une signature électronique auprès du service approprié. Un certificat authentifié peut être acheté auprès de sociétés reconnues, telles VeriSign (<http://www.verisign.com>) ou Thawte (<http://www.thawte.com>).

Vous pouvez également créer vos propres certificats numériques (non authentifiés) en utilisant l'utilitaire de certification fourni avec Microsoft Office. Ouvrez le menu Démarrer puis, dans la liste des programmes, choisissez Microsoft Office > Certificat numérique pour les projets VBA (voir Figure 16.13).

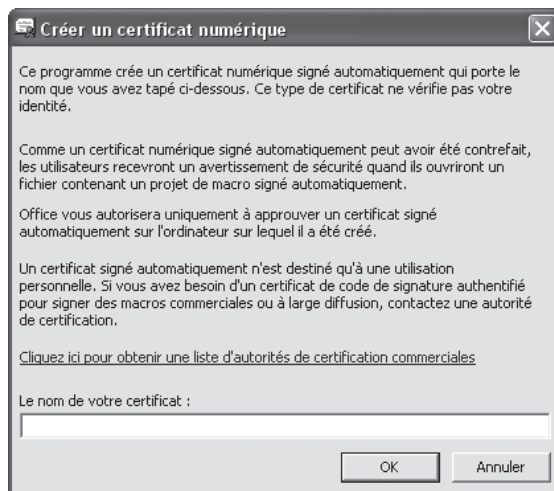


Si la commande Certificat numérique n'apparaît pas dans le groupe Microsoft Office, recherchez le fichier Selfcert.exe sur les CD-ROM d'Office.

Pour une liste exhaustive des organismes de certification, consultez l'aide en ligne d'Office ou, dans la boîte de dialogue représentée à la Figure 16.13, cliquez sur le lien.

Figure 16.13

Vous pouvez créer vos propres certificats numériques.



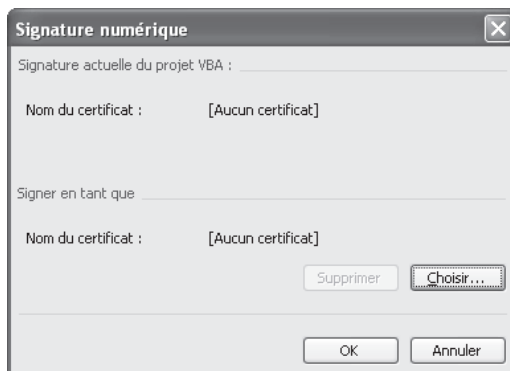
Authentifier une macro

Lorsque vous avez obtenu une signature électronique, il ne vous reste qu'à "signer" vos macros. Pour signer une macro, procédez comme suit :

1. Sélectionnez le projet que vous souhaitez authentifier dans l'Explorateur de projet.
2. Choisissez Outils > Signature électronique (voir Figure 16.14).
3. Cliquez sur le bouton Choisir et, dans la fenêtre qui s'affiche, choisissez la signature électronique que vous souhaitez affecter à votre projet.

Figure 16.14

La boîte de dialogue Signature numérique permet de gérer les signatures affectées à vos projets.





Exemple complet d'application Excel

Au sommaire de ce chapitre

- Présenter un projet d'application Excel
- Créer un modèle Excel
- Définir et créer des interfaces utilisateur
- Écrire des procédures d'édition de documents

Ce chapitre constitue un récapitulatif de l'ensemble des connaissances acquises au long de votre lecture. Nous vous proposons d'y créer un programme complet, étape par étape, de sa définition à son intégration à l'interface d'Excel, qui mette en pratique l'ensemble des connaissances acquises au cours de cet ouvrage.

Présenter un projet d'application Excel

Le programme que nous allons écrire ici aura pour fonction de générer des factures de droits d'auteur et de mettre à jour un tableau Word. La Figure 17.1 présente une facture type.

Afin de simplifier et de sécuriser cette tâche, nous créerons des interfaces utilisateur adaptées, dans lesquelles il suffira d'entrer les données nécessaires au programme. Le programme aura pour fonction de générer les feuilles de paie, de les enregistrer et d'en imprimer un nombre d'exemplaires défini par l'utilisateur. Il mettra également à jour un tableau dans un document Word répertoriant les informations essentielles concernant le contrat.

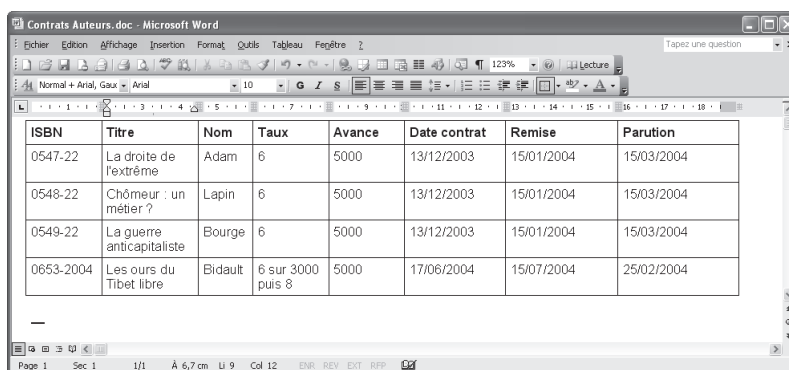
Figure 17.1

*Une facture
droits d'auteur type.*

	A	B	C	D	E	F	G
1	DROITS D'AUTEURS						
2	AUTEUR / SOCIETE		Mikaël Bidault				
3	TITRE		Les ours du tibet				
4	ISBN		111-71				
5							
6	DATE DE REMISE		25/12/2004				
7	DATE DE PARUTION		01/02/2005				
8							
9	ADRESSE		5 rue du soleil				
10	CODE POSTAL		75020				
11	VILLE		Paris				
12	PAYS		France				
13	DROITS D'AUTEUR						
14							
15	MONTANT BRUT A PAYER					2 500,00	
16	TVA (0,8%)					20,00	
17	AGESSA (0,85%)					21,25	
18	BASE CSG (95% MONTANT BRUT)					2 375,00	
19	CSG (7,50%)					178,13	
20	BASE RDS (95% MONTANT BRUT)					2 375,00	
21	RDS (0,5%)					11,88	
22							
23	NET A PAYER					2 308,75	
24							
25							
26	TVA A VERSER					117,50	
27	CONTRIBUTION DIFFUSEUR (1%)					25,00	
28							
29							
30	DATE DE REGLEMENT						
31							
32	DROITS D'AUTEUR	4000 premiers exemplaires				8% (huit pour cent)	
33		Au-delà				10% (dix pour cent)	
34							

Avant de commencer

Pour les besoins du programme, créez un fichier Word semblable à celui représenté à la Figure 17.2 et enregistrez-le sous le nom Contrats Auteurs.doc dans le dossier C:\Mes documents\.



ISBN	Titre	Nom	Taux	Avance	Date contrat	Remise	Parution
0547-22	La droite de l'extrême	Adam	6	5000	13/12/2003	15/01/2004	15/03/2004
0548-22	Chômeur : un métier ?	Lapin	6	5000	13/12/2003	15/01/2004	15/03/2004
0549-22	La guerre anticapitaliste	Bourge	6	5000	13/12/2003	15/01/2004	15/03/2004
0653-2004	Les ours du Tibet libre	Bidault	6 sur 3000 puis 8	5000	17/06/2004	15/07/2004	25/02/2004

Figure 17.2

Chaque fois qu'un contrat est édité, les données sont intégrées dans un fichier Word.

Le programme VBA que vous développerez dans ce chapitre utilise le contrôle Calendrier pour inviter l'utilisateur à sélectionner des dates. Celui-ci n'est pas activé par défaut, commencer par le référencer dans la boîte à outils de Visual Basic. La procédure est décrite à la section "Personnaliser la boîte à outils" du Chapitre 12.



Le contrôle calendrier n'est pas fourni avec la version 2010 d'Office. Il est cependant très simple de l'installer et de le référencer. Reportez-vous pour cela à la procédure susnommée.

Identification des informations à recueillir

Les informations nécessaires à l'établissement d'une feuille de paie de droits d'auteur sont les suivantes :

- **Auteur.** Nom et prénom (ou société), adresse, code postal, ville, pays (facultatif).

Les adresses des auteurs sont stockées dans un fichier Excel (voir Figure 17.3). Une interface permettra de sélectionner l'auteur dans la liste et mettra à jour son adresse sans qu'il soit nécessaire pour l'utilisateur de la connaître. Si l'auteur ne fait pas partie de la liste, l'utilisateur pourra saisir l'ensemble des coordonnées et les ajouter au fichier Excel.

Le contrat pourra être établi au nom d'une personne physique ou à celui d'une société. Dans le premier cas, un prénom devra être fourni.

Figure 17.3

La liste des auteurs est stockée dans un fichier Excel.

	A	B	C	D	E	F	G
	Nom	Prénom	Adresse	Ville	Code postal	Pays	
2	Bateau	Hervé	4, rue des Eléphants	Les Ours	67500		
3	Duboeuf	René	impasse des hirondelles	Madelon	78542		
4	Ducoq	Bertrand	10, avenue de la belle moselle	Bagny	78560		
5	François	Jean	4, rue de Belleville	Paris	75019		
6	Labague	Samule	2, rue du Soleil	Paris	75020		
7	Manon	Jean-François	17, place de l'église	Poullan-sur m	29000		
8	R. Munan	I. nic	25, allée des Quatre saisons	Ottignies	3485	Belgique	

Info

Pour les besoins du programme, créez un fichier Excel semblable à celui de la Figure 17.3 et enregistrez-le sous le nom Classeur Auteurs.xlsx dans le dossier C:\Mes documents.

- **Ouvrage.** Titre de l'ouvrage et ISBN (International Standard Book Number, numéro identifiant l'ouvrage de façon unique).

L'ISBN sera toujours formaté de la façon suivante : xxxx-y, où chaque x correspond à un chiffre, et chaque y à une clé pouvant être un chiffre ou une lettre.

- **Conditions de rémunération.** Droits d'auteur et avance sur droits d'auteur.

Les droits d'auteur sont un pourcentage du prix de vente de l'ouvrage. Ce pourcentage peut être fixe ou variable (par exemple, 6 % sur les 3 000 premiers livres, puis 8 %).

L'avance sur droits d'auteur est facultative. Dans le cas d'une avance, celle-ci pourra être versée en une fois (à la remise du manuscrit) ou en deux fois (une moitié à la remise et une moitié à la parution).

- **Taxes.** Les taxes retenues sur l'avance sur droits d'auteur sont la TVA, les AGESSA, la CSG et le RDS. Le programme n'aura cependant pas à les traiter, puisque nous créerons un classeur modèle dans lequel nous intégrerons les formules nécessaires.
- **Dates.** Date de remise et date de parution.

La date de parution devra toujours être postérieure à la date de remise. Par ailleurs, si moins de 40 jours séparent la date de remise de la date de parution, une confirmation sera demandée à l'utilisateur.

- **Options d'impression.** L'utilisateur pourra choisir d'imprimer ou non les documents édités. Il pourra également définir le nombre d'exemplaires imprimés pour le courrier et pour les feuilles de paie.

Définition de la structure du programme

Afin d'assurer une bonne lisibilité au programme et de pouvoir réutiliser les procédures dans d'autres conditions (par exemple, pour un programme équivalent concernant un travail de traduction), les procédures seront aussi autonomes qu'il se peut. Nous devons ici définir le squelette du programme (son flux), les modules de stockage des procédures et la façon dont seront stockées les informations fournies par l'utilisateur.

Le squelette du programme

Le programme sera structuré de la façon suivante :

1. Déclaration des variables objet et liaison avec les fichiers à lire ou à manipuler.
Il s'agit de créer les liaisons avec le répertoire des auteurs et le document Word à compléter.
2. Affichage des interfaces utilisateur.
On commence par se procurer l'ensemble des informations requises. Pour chaque interface :
 - a. Chargement des feuilles (procédures `Initialize`).
Les interfaces sont appelées à partir de la procédure principale.
 - b. Vérification des informations fournies par l'utilisateur.
Les procédures événementielles des feuilles vérifient la validité des données.
 - c. Stockage des données dans des variables.
Lorsque les données sont valides, les valeurs entrées par l'utilisateur sont affectées aux variables appropriées, afin de pouvoir être ensuite exploitées par le programme.
 - d. Masquage de l'interface.
3. Édition, impression éventuelle et enregistrement des feuilles de paie.
Un nouveau classeur fondé sur un modèle est créé. Les informations nécessaires sont entrées dans les cellules appropriées.

4. Mise à jour du tableau Word.

Une ligne est ajoutée au tableau Word et les données du contrat y sont intégrées.

5. Fin du programme et libération des ressources mémoire.

Créez un nouveau module de code et appelez-le `ContratAuteur`. Placez-y tout de suite le corps de la procédure principale, en plaçant sous forme de commentaires les phases principales :

```
Sub EditionContratAuteur()  
  
    '1. Liaisons des var. objet  
  
    '2. Affichage des feuilles  
  
    '3. Edition des feuilles de paie  
  
    '4. Mise à jour du tableau Word  
  
    '5. libération des ressources mémoire  
  
End Sub
```

Les modules

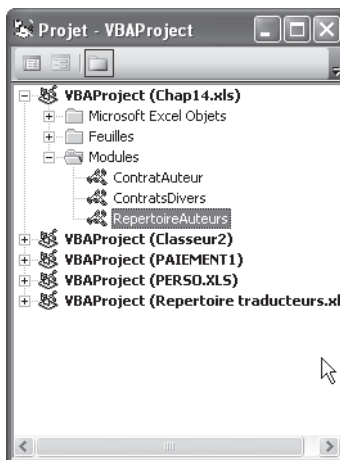
Les procédures seront regroupées par modules. Créez les modules suivants :

- **ContratAuteur.** On y retrouvera les déclarations de variables, la procédure principale et les procédures spécifiques au contrat d'auteur. Les variables devront être publiques afin de pouvoir être manipulées à partir de n'importe quel module, et notamment à partir des interfaces, afin de leur affecter les valeurs données entrées par l'utilisateur.
- **ContratsDivers.** Ce module regroupera les procédures susceptibles d'être réutilisées par d'autres programmes (par exemple un programme de contrat de traduction), telles que l'impression, l'enregistrement, etc.
- **RepertoireAuteurs.** Ce module regroupera les procédures destinées à manipuler le répertoire des auteurs. Les procédures devront être publiques pour pouvoir être appelées à partir du module `ContratAuteur` et manipulées à partir des interfaces utilisateur.

Le dossier Modules de l'Explorateur de projet doit être semblable à celui de la Figure 17.4.

Figure 17.4

L'Explorateur de projet à ce stade de la création du programme.



Le stockage des informations fournies par l'utilisateur

Les informations fournies par l'utilisateur au travers des interfaces qu'il sera invité à compléter seront stockées dans des types de données personnalisés afin de permettre un regroupement des informations de même type :

- Le type Auteur contiendra les données relatives à l'auteur (nom et adresse) : nom, prénom, adresse, code postal, ville, pays, société.

Le prénom, le nom, l'adresse, le code postal, la ville et le pays seront des valeurs de type String (chaîne de caractères). La société sera une valeur booléenne (contrat au nom d'une société = True, contrat au nom d'une personne physique = False).

- Le type ConditionsAuteur rassemblera les données relatives à l'ouvrage et aux droits d'auteur (titre, ISBN, taux variable ou non, taux initial des droits d'auteur sous forme numérique et sous forme de chaîne, nombre d'exemplaires affectés par le taux initial, taux au-delà de x exemplaires sous forme numérique et sous forme de chaîne, avance ou non, valeur de l'avance, nombre de paiements, date de remise et date de parution).

Le titre, l'ISBN et les taux sous forme de chaînes seront des valeurs de type String. Le taux initial et le taux à suivre, ainsi que le nombre de paiements sur lequel sera effectuée l'avance seront des valeurs numériques de type Byte (supérieures à 0 et inférieures à 256). Le nombre d'exemplaires sur lequel portera le taux initial et la valeur de l'avance éventuelle seront des valeurs numériques de type Integer. Le taux variable sera une valeur booléenne (taux variable = True, taux fixe = False). Avance ou non correspond également à une valeur booléenne (avance = True, pas d'avance = False). La date de remise et la date de parution seront des valeurs de type Date.

- Le type OptionsImpression rassemblera les informations concernant l'impression du document (imprimer ou non, imprimer ou non le courrier, imprimer ou non les feuilles de paie, nombre d'exemplaires du courrier à imprimer et nombre d'exemplaires des feuilles de paie à imprimer).

L'impression, l'impression du courrier et l'impression des feuilles de paie seront des valeurs numériques (Imprimer = True, Ne pas imprimer = False). Le nombre de copies du courrier et le nombre de copies des feuilles de paie seront des valeurs de type Variant.

Placez d'ores et déjà les déclarations de ces types en tête du module ContratAuteur :

'Types de données personnalisées

Type Auteur

```
Prenom As String
Nom As String
Adresse As String
CodePostal As String
Ville As String
Pays As String
societe As Boolean
```

End Type

Type ConditionsAuteur

```
Titre As String
ISBN As String
TauxVariable As Boolean
TauxDroitsNum1 As Byte
TauxDroitsNum2 As Byte
TauxDroitsStr1 As String
TauxDroitsStr2 As String
PourCent1Str As String
PourCent2Str As String
NumExemplairesTaux1 As Integer
AvanceOuNon As Boolean
AvanceSurDroits As Integer
NumPaievements As Byte
remise As Date
Parution As Date
```

End Type

Type OptionsImpression

```
Imprimer As Boolean
ImprimerCourrier As Boolean
```

```

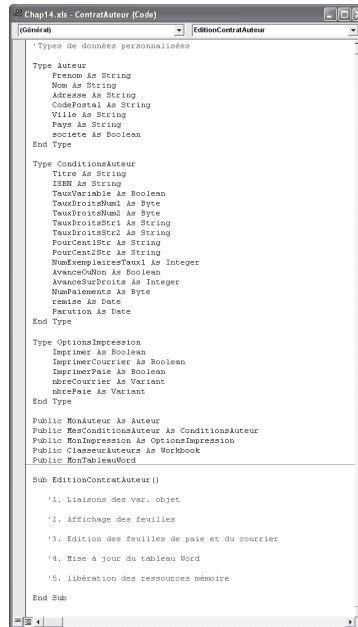
ImprimerPaie As Boolean
nbreCourrier As Variant
nbrePaie As Variant
End Type

```

La Figure 17.5 présente le module ContratAuteur tel qu'il doit se présenter à ce stade.

Figure 17.5

Le module ContratAuteur contient les déclarations de types et le corps de la procédure principale.



Ajoutez sous les déclarations de type, la déclaration des variables publiques dans lesquelles seront stockées les données. Ces déclarations doivent se trouver dans la zone de déclarations du module (hors de toute procédure) pour être accessible à partir de n'importe quel module du projet.

```

Public MonAuteur As Auteur
Public MesConditionsAuteur As ConditionsAuteur
Public MonImpression As OptionsImpression
Public ClasseurAuteurs As Workbook
Public MonTableauWord

```

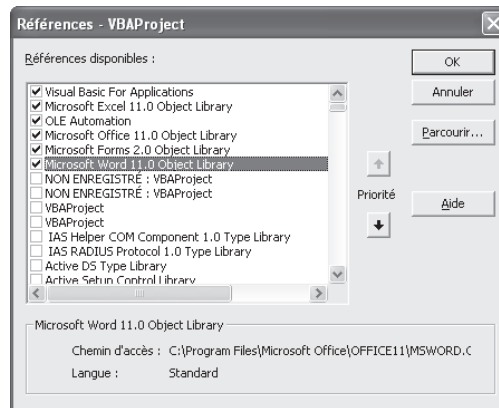
Les trois premières instructions déclarent les variables dans lesquelles seront stockées les informations fournies par l'utilisateur. Les variables ClasseurAuteurs et MonTableauWord se verront respectivement affecter le classeur des auteurs et le document Word contenant le tableau récapitulatif des contrats.



Pour pouvoir manipuler des fichiers Word, vous devez créer une référence à la bibliothèque d'objets de Word. Choisissez la commande Références du menu Outils de Visual Basic Editor et cochez la case Microsoft Word Object Library (voir Figure 17.6). Si vous omettez de le faire, l'instruction de déclaration de la variable `MonTableauWord` générera une erreur.

Figure 17.6

Créez une référence vers la bibliothèque d'objets de Word.



Ajoutez les liaisons des variables objet dans la procédure `EditionContratAuteur`.

```
Sub EditionContratAuteur()

    '1. Liaisons des var. objet
    Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
    Set MonTableauWord = GetObject(, "Word.Application")
    If Err.Number <> 0 Then Err.Clear

    '2. Affichage des feuilles

    '3. Edition des feuilles de paie et du courrier

    '4. Mise à jour du tableau Word

    '5. libération des ressources mémoire

End Sub
```



Veillez à personnaliser les chemins permettant l'accès aux fichiers utilisés dans cet exemple.

Créer un modèle Excel

Nous allons maintenant créer le modèle Excel qu'utilisera le programme pour générer les feuilles de paie.

Créez un nouveau classeur Excel. Supprimez les feuilles 2 et 3, et renommez la feuille 1 en "Auteur". Entrez ensuite les données fixes, puis les formules comme indiqué à la Figure 17.7. Formatez les cellules (bordures, polices, formats, etc.). Définissez un format numérique à deux décimales pour les cellules devant recevoir une valeur monétaire, et un format de date pour les cellules devant recevoir des données de type date. Enregistrez ensuite le classeur en tant que modèle (extension .xlt), sous le nom FeuillePaieAuteur, puis fermez-le.

Figure 17.7

Le modèle
FeuillePaieAuteur
complété.

		=F20*0,5%	=F18*7,5%	=F15*0,85%	=F15*0,8%	=F15*95%	
	A	B	C	D	E	F	G
1	DROITS D'AUTEURS						
2	AUTEUR / SOCIETE						
3	TITRE						
4	ISBN						
5							
6	DATE DE REMISE						
7	DATE DE PARUTION						
8							
9	ADRESSE						
10	CODE POSTAL						
11	VILLE						
12	PAYS						
13	DROITS D'AUTEUR						
14							
15	MONTANT BRUT A PAYER						
16	TVA (0,8%)					0,00	
17	AGESSA (0,85%)					0,00	
18	BASE CSG (95% MONTANT BRUT)					0,00	
19	CSG (7,50%)					0,00	
20	BASE RDS (95% MONTANT BRUT)					0,00	
21	RDS (0,5%)					0,00	
22							
23	NET A PAYER					0,00	
24							
25							
26	TVA A VERSER					0,00	
27	CONTRIBUTION DIFFUSEUR (1%)					0,00	
28							
29							
30	DATE DE REGLEMENT						
31							
32	DROITS D'AUTEUR						
33							
34							
=F15+F16 F17 F19 F21			=F15*4,7%			=F15*1%	=F18

Conseil

Testez votre feuille de calcul. Entrez une valeur dans la cellule F15 et voyez si la valeur des cellules liées est mise à jour avec les valeurs appropriées.

Nous n'entrerons pas ici dans le détail des formules et la façon de calculer les droits d'auteur. Notez simplement que la CSG et le RDS sont calculés sur une base de 95 % du montant brut, et que le montant net se calcule de la façon suivante :

$$\text{Net} = \text{Brut} + \text{TVA} - \text{AGESSA} - \text{CSG} - \text{RDS}$$

La "Contribution à verser" et la TVA (lignes 26 et 27) sont des taxes dues par l'éditeur.

Le Tableau 17.1 présente les informations qui devront être complétées par le programme dans la feuille de paie.

Tableau 17.1 : Définition des cellules qui devront être renseignées par le programme

<i>Cellule</i>	<i>Contenu</i>
C2	Auteur (Prénom + Nom) ou Société
C3	Titre de l'ouvrage
C4	ISBN
C6	Date de remise
C7	Date de parution
C9	Adresse
C10	Code postal
C11	Ville
C12	Pays
F15	Montant de l'avance sur droits à payer
F30	Date de règlement
B32	Nombre d'exemplaires touchés par le taux 1
F32	Taux 1 sous forme de valeur numérique
F33	Taux 2 sous forme de valeur numérique
G32	Taux 1 sous forme de chaîne
G33	Taux 2 sous forme de chaîne

Définir et créer des interfaces

Le programme proposera 5 interfaces qui permettront de rassembler les différentes informations requises pour éditer le contrat :

- **fmContratAuteur.** Cette feuille permettra de choisir un auteur dans la liste des auteurs (extraite du fichier Excel).
- **fmContratConditions.** Cette feuille permettra de fournir les informations concernant l'ouvrage (titre et ISBN) et les conditions du contrat (taux des droits d'auteur et avance).
- **fmContratDates.** Cette feuille permettra de préciser la date de remise et la date de parution.
- **fmContratImpression.** Cette feuille permettra de définir les documents à imprimer et le nombre d'exemplaires.
- **fmContratFin.** Cette feuille indiquera à l'utilisateur que les informations nécessaires à l'édition du contrat ont été recueillies.

Toutes les feuilles, à l'exception de la première, proposeront un bouton de commande permettant de réafficher la feuille précédente. Toutes les feuilles contiendront un bouton de validation et un bouton d'Annulation.

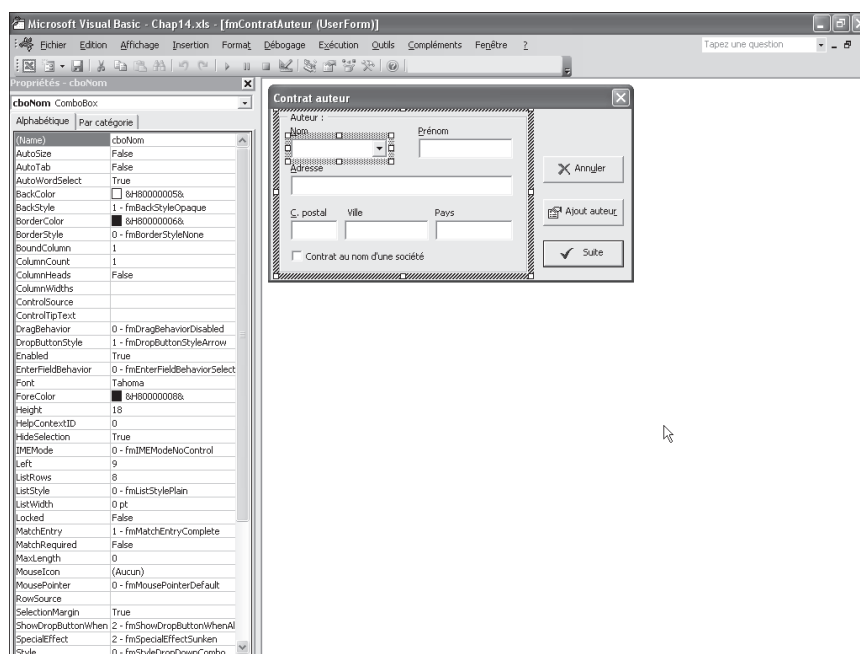
Feuille *fmContratAuteur*

La feuille *fmContratAuteur* contiendra une zone de liste modifiable permettant de sélectionner le nom de l'auteur dans la liste des auteurs (stockée dans le classeur Excel Répertoire Auteurs.xlsx) qui sera chargée à l'affichage de la feuille. Elle contiendra également cinq zones de texte qui permettront respectivement d'indiquer le prénom, l'adresse, le nom, le code postal, la ville et le pays. Une case à cocher permettra de préciser s'il s'agit d'une société. Outre les boutons Suite et Annuler, un bouton Ajouter Auteur permettra d'ajouter les coordonnées d'une personne au fichier Excel des auteurs. La Figure 17.8 présente la feuille *fmContratAuteur* en mode Conception.



Afin de mettre en valeur les différentes phases d'écriture des procédures événementielles d'une feuille, nous détaillerons les procédures de la feuille fmContratAuteur pour chaque contrôle. Pour les feuilles suivantes, nous présenterons l'ensemble des procédures en un seul listing que nous commenterons.

Créez la feuille *fmContratAuteur* en vous fondant sur la Figure 17.8. Placez un contrôle Frame sur la feuille et, à l'intérieur de celui-ci, un contrôle ComboBox, cinq contrôles TextBox et un contrôle CheckBox. Placez un contrôle Label au-dessus de chaque contrôle TextBox et au-dessus du contrôle ComboBox afin d'en libeller la fonction (Nom, Prénom, Adresse, Code postal, Ville, Pays).

**Figure 17.8**

La feuille fmContratAuteur en mode Conception.

Placez trois contrôles CommandButton hors du contrôle Frame. Affectez les propriétés suivantes aux contrôles :

Propriété	Valeur
Feuille	
Name	fmContratAuteur
Caption	Contrat auteur
Contrôle Frame	
Caption	Auteur :
Contrôle ComboBox	
Name	cboNom
Style	0- fmStyleDropDownCombo
MatchEntry	1 - fmMatchEntryComplete
Contrôle TextBox 1	
Name	txtPrenom

<i>Propriété</i>	<i>Valeur</i>
Contrôle TextBox 2	
Name	txtAdresse
Contrôle TextBox 3	
Name	txtCodePostal
Contrôle TextBox 4	
Name	txtVille
Contrôle TextBox 5	
Name	txtPays
Contrôle Label 1	
Name	lbNom
Caption	Nom
Contrôle Label 2 ¹	
Name	lbPrenom
Caption	Prénom
Contrôle CheckBox	
Name	chkSociete
Caption	Contrat au nom d'une société
Contrôle CommandButton 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle CommandButton 2	
Name	CmdAjouterAuteur
Caption	Ajouter auteur
Contrôle CommandButton 3	
Name	CmdSuite
Caption	Suite
Default	True

1. Il n'est pas nécessaire de définir les propriétés Name des autres contrôles Label, car ils ne seront pas manipulés.

Ouvrez ensuite la fenêtre Code de la feuille et placez-y les procédures suivantes :

1. Code d'initialisation de la feuille

```
1: Private Sub UserForm_Initialize()  
2:     Application.StatusBar = «Chargement des auteurs en cours.  
   Veuillez patienter...»  
3:     Application.Cursor = xlWait  
4:     Call MiseAJourListeDeroulante  
5:     Application.StatusBar = «»  
6:     Application.Cursor = xlDefault  
7:     cboNom.SetFocus  
8: End Sub  
9:  
10: Private Sub MiseAJourListeDeroulante()  
11:     'Suppression des entrées de la liste si celle-ci en contient  
12:     If cboNom.ListCount >= 1 Then  
13:         Dim ElementListe As Integer  
14:         Dim NbreElt As Integer  
15:         NbreElt = cboNom.ListCount - 1  
16:         For ElementListe = NbreElt To 0 Step -1  
17:             cboNom.RemoveItem (ElementListe)  
18:         Next ElementListe  
19:     End If  
20:     'Ajout de tous les noms du repertoire des auteurs  
21:     Dim compteur As Long  
22:     Dim AjoutAuteur As String  
23:     For compteur = 2 To ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row  
24:         AjoutAuteur = ClasseurAuteurs.Sheets(1).Range("A» & compteur).Value  
25:         cboNom.AddItem (AjoutAuteur)  
26:     Next compteur  
27: End Sub
```

La procédure d'initialisation commence par afficher un message dans la barre d'état de l'application afin d'informer l'utilisateur de la procédure en cours. Ligne 3, le curseur est transformé en sablier. Ligne 4, la procédure `MiseAJourListeDeroulante` est appelée. Après que celle-ci s'est exécutée, la procédure appelante reprend la main, et le message de la barre d'état est effacé tandis que le curseur reprend sa forme normale. Enfin, ligne 7, la zone de liste modifiable reçoit le focus.



La variable `ClasseurAuteurs` doit avoir été déclarée et un classeur doit lui être affecté avant que la feuille ne soit affichée. Si tel n'est pas le cas, l'instruction de la ligne 23 générera une erreur au moment de l'affichage de la feuille.

La procédure `MiseAJourListeDeroulante` a pour fonction d'ajouter les noms des auteurs à la zone de liste modifiable. Les instructions des lignes 11 à 19 suppriment les éléments de la liste si celle-ci n'est pas vide. Nous y reviendrons plus tard. Lignes 20 à 26, les auteurs sont ajoutés à la liste. On utilise pour cela une structure `For...Next` qui ajoute un à un le contenu des cellules de la colonne A contenant des données à la liste déroulante : un compteur démarre à 2 (la cellule A2 étant la première à contenir un nom d'auteur) pour atteindre le numéro de ligne correspondant à la dernière cellule non vide de la colonne A [`(Range("A1").End(xlDown).Row)`].



À ce stade, vérifiez que la mise à jour de la zone de liste modifiable s'effectue correctement. Exécutez pour cela la procédure `EditionContratAuteur`, après avoir pris soin d'y ajouter l'instruction d'affichage de la feuille. Celle-ci doit se présenter comme suit :

```
Sub EditionContratAuteur()
    Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur
Auteurs.xlsx")
    Set MonTableauWord = GetObject(, "Word.Application")
    If Err.Number <> 0 Then Err.Clear
    FmContratAuteur.Show
End Sub
```

2. Code de mise à jour des zones de texte

```
1: Private Sub cboNom_Change()
2:     Dim LigneSel As Long
3:     LigneSel = cboNom.ListIndex + 2
4:     txtPrenom = ClasseurAuteurs.Sheets(1).Range("B" & LigneSel).Value
5:     txtAdresse = ClasseurAuteurs.Sheets(1).Range("C" & LigneSel).Value
6:     TxtVille = ClasseurAuteurs.Sheets(1).Range("D" & LigneSel).Value
7:     TxtCodePostal = ClasseurAuteurs.Sheets(1).Range("E" & LigneSel).Value
8:     TxtPays = ClasseurAuteurs.Sheets(1).Range("F" & LigneSel).Value
9: End Sub
```

Cette procédure événementielle met automatiquement à jour le contenu des zones de texte lorsque l'utilisateur modifie le contenu de la zone de liste modifiable Nom (par la sélection d'un nom dans la liste ou par la saisie d'un nouveau nom). La variable `LigneSel` sert

à stocker le numéro de ligne du classeur des auteurs correspondant au nom sélectionné. La propriété `ListIndex` du contrôle `cboNom` renvoie l'index de l'élément sélectionné. On y ajoute 2 afin d'obtenir le numéro de la ligne correspondante dans le classeur Excel – le premier nom de la liste correspond à la ligne 2 et la première valeur d'index d'un contrôle `ComboBox` est 0.

Lignes 4 à 8, la valeur de chacun des contrôles `TextBox` est mise à jour et reçoit pour valeur le contenu de la cellule située sur la même ligne que le nom sélectionné et dans la colonne correspondant au contrôle. Par exemple, si l'utilisateur sélectionne un nom provenant de la cellule A10, la zone `txtPrenom` reçoit le contenu de la cellule B10, la zone `txtAdresse` reçoit le contenu de la valeur C10, etc.

Notez que si l'utilisateur saisit une valeur dans le contrôle `ComboBox` (ce qui est possible, car la propriété `Style` du contrôle a été définie à 0 - `fmStyleDropDownCombo`), le complément automatique apparaît si un nom correspondant aux premières lettres saisies existe dans la liste (car la propriété `MatchEntry` a été définie à 1 - `fmMatchEntryComplete`) et que les zones de texte sont automatiquement mises à jour avec les valeurs correspondant à ce nom. Si l'utilisateur saisit un nom qui ne fait pas partie de la liste, la propriété `ListIndex` du contrôle renvoie la valeur - 1. `LigneSel` reçoit alors la valeur 1 (- 1 + 2) et les zones de texte affichent donc le contenu des cellules de la ligne 1, soit le titre des colonnes (voir Figure 17.9).

Figure 17.9

Lorsque l'utilisateur sélectionne un nom dans la liste ou saisit un nouveau nom, le complément automatique est proposé et les zones de texte sont mises à jour.

The figure consists of two screenshots of a VBA dialog box titled "Contrat auteur".

Top Screenshot:

- Auteur :**
 - Nom:** A dropdown menu showing "Biteau".
 - Prénom:** A text box containing "Hervé".
- Adresse:** A text box containing "4, rue des Eléphants".
- C. postal:** A text box containing "67500".
- Ville:** A text box containing "Les Ours".
- Pays:** A text box containing "Pays".
- Contrat au nom d'une société:** An unchecked checkbox.
- Buttons:** "Annuler" (with a close icon), "Ajout auteur" (with a plus icon), and "Suite" (with a checkmark icon).

Bottom Screenshot:

- Auteur :**
 - Nom:** A dropdown menu showing "Bidault".
 - Prénom:** A text box containing "Prénom".
- Adresse:** A text box containing "Adresse".
- C. postal:** A text box containing "Code po".
- Ville:** A text box containing "Ville".
- Pays:** A text box containing "Pays".
- Contrat au nom d'une société:** An unchecked checkbox.
- Buttons:** "Annuler" (with a close icon), "Ajout auteur" (with a plus icon), and "Suite" (with a checkmark icon).

3. Code de la case à cocher Société

```
1: Private Sub ChkSociete_Click()  
2:   If ChkSociete.Value = True Then  
3:     LbNom.Caption = "Société"  
4:     LbPrenom.Visible = False  
5:     txtPrenom.Visible = False  
6:   Else  
7:     LbNom.Caption = "Nom"  
8:     LbPrenom.Visible = True  
9:     txtPrenom.Visible = True  
10:  End If  
11: End Sub
```

La case à cocher Société est cochée si le contrat est établi au nom d'une société. Cette procédure gère l'événement Click du contrôle. Si la case est cochée (ligne 2), le contrôle Label lbNom voit sa propriété Caption redéfinie à Société et les contrôles lbPrenom et lbNom sont masqués (Visible = False). Si la case est décochée, la propriété Caption du contrôle lbNom est redéfinie à Nom et les propriétés Visible des contrôles lbPrenom et txtPrenom sont définies à True de façon à les afficher (voir Figure 17.10).

Figure 17.10

Les libellés Nom et Prénom, ainsi que la zone de texte Prénom sont affectés par l'état de la case à cocher Société.

The figure consists of two screenshots of a Windows-style dialog box titled "Contrat auteur".

Top Screenshot: The "Auteur :" section has a "Nom" label and a "Prénom" label. Below "Nom" is a dropdown menu showing "Bidault". Below "Prénom" is a text box labeled "Prénom". The "Adresse" section has a text box labeled "Adresse". Below that are three text boxes: "C. postal", "Ville", and "Pays". The "Code po:" label is to the left of the "Ville" box. At the bottom left, the checkbox "Contrat au nom d'une société" is unchecked. On the right, there are three buttons: "Annuler", "Ajout auteur", and "Suite".

Bottom Screenshot: The "Auteur :" section has a "Société" label. Below it is the same dropdown menu showing "Bidault". The "Prénom" label and its text box are no longer visible. The "Adresse" section and the "C. postal", "Ville", "Pays" text boxes remain visible. The "Code po:" label is still to the left of the "Ville" box. At the bottom left, the checkbox "Contrat au nom d'une société" is now checked. The buttons on the right remain the same.

4. Code d'annulation

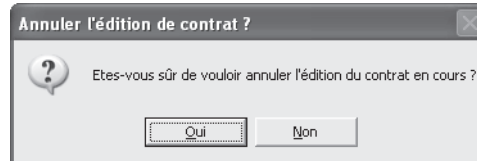
```
1: Private Sub cmdAnnuler_Click()
2:   Dim rep As Byte
3:   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
   cours", _
4:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
5:
6:   If rep = vbYes Then
7:     Me.Hide
8:     Call ContratFin
9:   End
10: End If
11: End Sub
12:
13: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
14:   Dim rep As Byte
15:   rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
   cours ?", _
16:     vbYesNo + vbQuestion, «Annuler l'édition de contrat ?»)
17:
18:   If rep = vbYes Then
19:     Me.Hide
20:     Call ContratFin
21:   End
22: End If
23: End Sub
24:
25: Public Sub ContratFin()
26: 'Libérer les ressources mémoire occupées par la var. objet
27: Set ClasseurAuteurs = Nothing
28: 'Curseur classique
29: Application.Cursor = xlDefault
30: End Sub
```

La procédure `cmdAnnuler_Click` gère l'annulation *via* le bouton Annuler – qui peut être liée à un clic sur le bouton ou à la frappe de la touche Échap (la propriété `Cancel` du contrôle est définie à `True`) –, tandis que la procédure `UserForm_QueryClose` gère un clic sur la case de fermeture de la fenêtre. Elles sont toutes deux semblables : la boîte de dialogue représentée à la Figure 17.11 s'affiche. Si l'utilisateur clique sur le bouton Oui, la procédure `ContratFin` est appelée et l'instruction `End` met fin au programme.

Placez la procédure `ContratFin` dans le module `ContratsDivers`. Elle est déclarée publique de façon à pouvoir être appelée à partir de n'importe quel module du projet. Elle a pour fonction de libérer la variable objet `ClasseurAuteurs` et de redéfinir le curseur.

Figure 17.11

Demandez toujours une confirmation afin d'éviter une annulation du programme par erreur.



5. Code du bouton Ajouter Auteur

```

1: Private Sub CmdAjouterAuteur_Click()
2:
3: 'Contrôle de la liste des auteurs
4:   Dim Ligne As Integer
5:   Application.StatusBar = «Contrôle de la liste des auteurs en cours.
   Veuillez patienter...»
6:   Application.Cursor = xlWait
7:   For Ligne = 2 To ClasseurAuteurs.Sheets(1).Range("A1").End(xlDown).Row
8:     If cboNom.Value = ClasseurAuteurs.Sheets(1).Range("A" & Ligne).Value
       And txtPrenom = ClasseurAuteurs.Sheets(1).Range("B" & Ligne).Value Then
9:       Application.StatusBar = «»
10:      Application.Cursor = xlDefault
11:      Dim remplacer As Integer
12:      remplacer = MsgBox("Ce prénom et ce nom existent déjà pour un
       auteur. Remplacer par les coordonnées actuelles ?",
       vbOKCancel + vbCritical, "Remplacer les coordonnées de l'auteur ?")
13:      If remplacer = vbCancel Then
14:        Exit Sub
15:      Else
16:        Application.StatusBar = «Remplacement des coordonnées et mise
       à jour en cours. Patientez...»
17:        Application.Cursor = xlWait
18:        Call RemplacerEntreeAuteur(Ligne, cboNom.Value, txtPrenom.
       Value, txtAdresse.Value, TxtVille.Value, TxtCodePostal.Value,
       TxtPays.Value)
19:        Application.StatusBar = ""
20:        Application.Cursor = xlDefault
21:        Exit Sub
22:      End If

```

```
23:         End If
24:     Next Ligne
25:     Application.StatusBar = «»
26:     Application.Cursor = xlDefault
27:
28: 'Ajout du nouvel auteur
29:     Application.StatusBar = «Ajout de l'auteur en cours. Veuillez patienter...»
30:     Application.Cursor = xlWait
31:     Call CreerEntreeAuteur(cboNom.Value, txtPrenom.Value, txtAdresse.Value,
        TxtVille.Value, TxtCodePostal.Value, TxtPays.Value)
32:
33: 'tri et enregistrement du repertoire
34:     ClasseurAuteurs.Sheets(1).Range(«A2:H» & ClasseurAuteurs.Sheets(1).
        Range(«A1»).End(xlDown).Row + 1).Sort Key1:=ClasseurAuteurs.Sheets(1).
        Range(«A2»), Key2:=ClasseurAuteurs.Sheets(1).Range(«B2»),
        Key3:=ClasseurAuteurs.Sheets(1).Range(«C2»)
35:     ClasseurAuteurs.Save
36:
37: 'Mise à jour de la liste déroulante
38:     StatusBar = «Mise à jour de la liste en cours. Veuillez patienter...»
39:     Call MiseAJourListeDeroulante
40:     Application.StatusBar = ""
41:     Application.Cursor = xlDefault
42: End Sub
43:
44:
45: Public Sub CreerEntreeAuteur(Nom As String, Prenom As String, Adresse As
String, _
46: Ville As String, CodePostal As String, Optional Pays As String = "")
47: Dim LigneAjout As Integer
48: LigneAjout = ClasseurAuteurs.Sheets(1).Range(«A1»).End(xlDown).Row + 1
49:
50: ClasseurAuteurs.Sheets(1).Range(«A» & LigneAjout).Value = Nom
51: ClasseurAuteurs.Sheets(1).Range(«B» & LigneAjout).Value = Prenom
52: ClasseurAuteurs.Sheets(1).Range(«C» & LigneAjout).Value = Adresse
53: ClasseurAuteurs.Sheets(1).Range(«D» & LigneAjout).Value = Ville
54: ClasseurAuteurs.Sheets(1).Range(«E» & LigneAjout).Value = CodePostal
55: ClasseurAuteurs.Sheets(1).Range(«F» & LigneAjout).Value = Pays
56: End Sub
57:
58: Public Sub RemplacerEntreeAuteur(Ligne As Integer, Nom As String, Prenom As
String, Adresse As String, _
59: Ville As String, CodePostal As String, Optional Pays As String = "")
```

```

60:
61: ClasseurAuteurs.Sheets(1).Range("A" & Ligne).Value = Nom
62: ClasseurAuteurs.Sheets(1).Range("B" & Ligne).Value = Prenom
63: ClasseurAuteurs.Sheets(1).Range("C" & Ligne).Value = Adresse
64: ClasseurAuteurs.Sheets(1).Range("D" & Ligne).Value = Ville
65: ClasseurAuteurs.Sheets(1).Range("E" & Ligne).Value = CodePostal
66: ClasseurAuteurs.Sheets(1).Range("F" & Ligne).Value = Pays
67:
68: ClasseurAuteurs.Sheets(1).Range("A2:H" & ClasseurAuteurs.Sheets(1).
    Range("A1").End(xlDown).Row).Sort Key1:=ClasseurAuteurs.Sheets(1).
    Range("A2"), _
69:     Key2:=ClasseurAuteurs.Sheets(1).Range("B2"), _
70:     Key3:=ClasseurAuteurs.Sheets(1).Range("C2")
71:
72: ClasseurAuteurs.Save
73: End Sub

```

La procédure `CmdAjouterAuteur_Click` (lignes 1 à 42) fait appel à deux procédures publiques : `CreerEntreeAuteur` (lignes 45 à 56) et `RemplacerEntreeAuteur` (lignes 58 à 73). Celles-ci seront stockées dans le module `RepertoireAuteurs` puisque leur fonction est de manipuler le fichier Excel des auteurs.

La procédure `CmdAjouterAuteur_Click` commence par vérifier s'il existe déjà une entrée dans le classeur avec le même nom et la même adresse (lignes 3 à 24). En fonction du résultat du contrôle l'une des deux procédures `CreerEntreeAuteur` ou `RemplacerEntreeAuteur` est appelée.

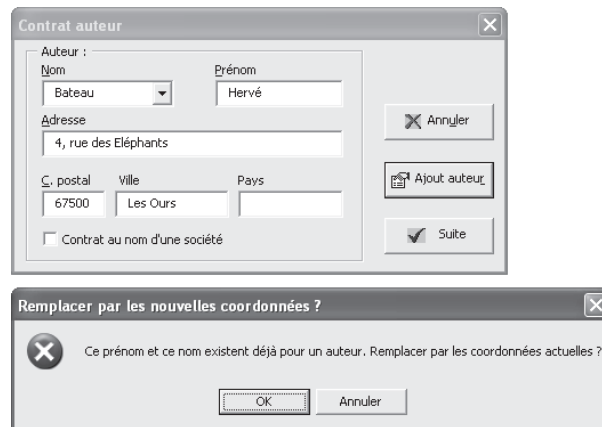
Le contrôle s'effectue à l'aide d'une boucle `For...Next` (lignes 7 à 24) qui contrôle les entrées du classeur de la ligne 2 à la dernière ligne contenant des données. Le curseur est préalablement transformé en sablier et un message s'affiche dans la barre d'état (lignes 5 et 6). À chaque passage de la boucle, le programme contrôle si les colonnes A et B de la ligne testée n'ont pas les mêmes valeurs que les noms et prénoms fournis dans les contrôles `cboNom` et `txtPrenom` de l'interface (ligne 8). Si tel est le cas, la barre d'état et le curseur retrouvent leurs valeurs par défaut et l'utilisateur se voit proposer le remplacement des coordonnées de l'auteur par celles entrées dans la feuille (ligne 12), comme présenté à la Figure 17.12. Si l'utilisateur choisit le bouton `Annuler`, l'instruction `Exit` de la ligne 14 entraîne la sortie de la procédure. S'il répond `Oui`, un message est à nouveau affiché dans la barre d'état et le curseur prend la forme d'un sablier (lignes 16 et 17). La procédure `RemplacerEntreeAuteur` est alors appelée. Lorsque la procédure appelante reprend la main, la barre d'état et le curseur reprennent leurs valeurs par défaut, puis une instruction `Exit` entraîne la sortie de la procédure.

Si aucune entrée semblable n'a été trouvée, le programme atteint la ligne 25 sans que rien ne se soit passé. Le nouvel auteur est alors ajouté sans qu'une intervention de l'utilisateur

ne soit requise. La procédure `CreerEntreeAuteur` est appelée ligne 31. Une fois l'auteur ajouté, la procédure appelante reprend la main et le classeur est trié et sauvegardé (lignes 34 et 35). La zone de liste modifiable est ensuite mise à jour (ligne 39). La procédure `MiseAJourListeDeroulante` qui prend en charge la mise à jour de liste déroulante a été présentée plus haut (voir la section «1. Code d'initialisation de la feuille»). Notez que cette procédure commence par supprimer les entrées de la liste pour les ajouter à nouveau. Enfin, lignes 40 et 41, la barre d'état et le curseur retrouvent leur aspect par défaut.

Figure 17.12

Le programme vérifie qu'il n'existe pas déjà une entrée pour l'auteur.



Les procédures `CreerEntreeAuteur` et `RemplacerEntreeAuteur` reçoivent toutes les deux les nom, prénom, adresse, code postal, ville et pays entrés par l'utilisateur dans l'interface. La procédure `RemplacerEntreeAuteur` reçoit en plus le numéro de la ligne à remplacer (notez les appels, lignes 18 et 31, et les déclarations, lignes 45 et 58). Tandis que la procédure `CreerEntreeAuteur` ajoute le nouvel auteur sur une ligne vide, la procédure `RemplacerEntreeAuteur` remplace les données de la ligne dont le numéro lui a été passé par celles fournies par l'utilisateur.

6. Code du bouton Suite

```

1: Private Sub CmdSuite_Click()
2:
3: 'Vérifier que les champs ont été correctement informés
4:   If cboNom.Value = «» Then
5:     MsgBox "Vous devez indiquer un nom d'auteur.", _
6:       vbOKOnly + vbInformation, "Informations incomplètes"
7:     cboNom.SetFocus
8:     Exit Sub
9:   ElseIf txtPrenom.Value = «» And ChkSociete.Value = False Then

```

```

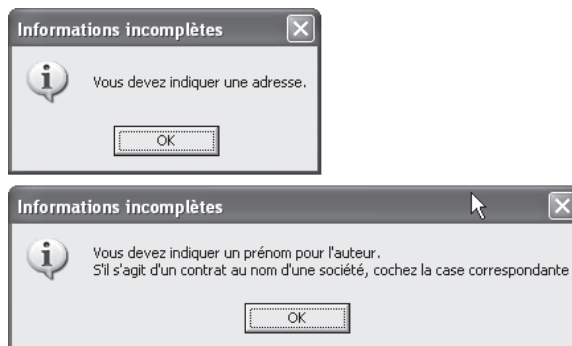
10:     MsgBox «Vous devez indiquer un prénom pour l'auteur.» & Chr(13) & _
11:     «S'il s'agit d'un contrat au nom d'une société, cochez la case
        correspondante», _
12:     vbOKOnly + vbInformation, «Informations incomplètes»
13:     txtPrenom.SetFocus
14:     Exit Sub
15: ElseIf txtAdresse.Value = "" Then
16:     MsgBox «Vous devez indiquer une adresse.», _
17:     vbOKOnly + vbInformation, «Informations incomplètes»
18:     txtAdresse.SetFocus
19:     Exit Sub
20: ElseIf TxtCodePostal.Value = "" Then
21:     MsgBox «Vous devez indiquer code postal.», _
22:     vbOKOnly + vbInformation, «Informations incomplètes»
23:     TxtCodePostal.SetFocus
24:     Exit Sub
25: ElseIf TxtVille.Value = "" Then
26:     MsgBox «Vous devez indiquer une ville.», _
27:     vbOKOnly + vbInformation, «Informations incomplètes»
28:     TxtVille.SetFocus
29:     Exit Sub
30: End If
31:
32: 'stockage des données dans des variables
33: With MonAuteur
34:     .Nom = cboNom.Value
35:     .Prenom = txtPrenom.Value
36:     .Adresse = txtAdresse.Value
37:     .CodePostal = TxtCodePostal.Value
38:     .Ville = TxtVille.Value
39:     If TxtPays.Value <> "" Then
40:         .Pays = Chr(13) + TxtPays.Value
41:     End If
42:     .societe = ChkSociete.Value
43: End With
44: Me.Hide
45: End Sub

```

La procédure `CmdSuite_Click` vérifie la validité des informations fournies par l'utilisateur et affecte ces valeurs aux variables appropriées. Lignes 3 à 30, une instruction conditionnelle est utilisée pour vérifier que chaque zone de texte a été complétée. Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur (voir Figure 17.13), le focus est passé au contrôle non renseigné, et une instruction `Exit` entraîne la sortie de la procédure. Notez l'expression conditionnelle de la ligne 9 qui vérifie si le prénom n'est pas fourni ET si la case Société n'est pas cochée.

Figure 17.13

Les données sont vérifiées avant d'être affectées aux variables.



Si les données fournies sont valides, la structure `With...End With` des lignes 33 à 43 affecte les valeurs fournies aux espaces de stockage appropriés de la variable publique `MonAuteur`. Notez que si le champ `pays` n'est pas renseigné (s'il s'agit de la France), l'espace `MonAuteur.Pays` ne reçoit pas de valeur. S'il est renseigné, il reçoit la valeur fournie précédée d'un retour chariot. Vous verrez plus tard pourquoi. Ligne 44, la feuille est masquée.

Feuille *fmContratConditions*

La feuille `fmContratConditions` contiendra un contrôle `Frame` libellé `Ouvrage` et un autre libellé `Conditions de rémunération`. Le cadre `Ouvrage` contiendra deux zones de texte permettant d'entrer le nom de l'ouvrage et son ISBN. Un contrôle `label` libellera chacune de ces zones de texte. Le cadre `Conditions de rémunération` contiendra deux autres cadres, l'un libellé `Droits d'auteur` et l'autre libellé `Avance`. Le cadre `Droits d'auteur` contiendra les contrôles permettant de préciser si le taux est variable et quels sont le taux initial et l'éventuel deuxième taux, et au-delà de combien d'exemplaires celui-ci s'applique. Le cadre `Avance` devra permettre de préciser s'il y a ou non une avance sur droits d'auteur et, dans l'affirmative, quelle est la valeur de cette avance. Enfin, l'utilisateur devra préciser si l'avance sera opérée en un ou en deux versements. Outre les boutons `Annuler` et `Suite`, un bouton `Retour` permettra de revenir à la feuille précédente.

La Figure 17.14 présente la feuille `fmContratConditions` en mode Conception.

Créez la feuille `fmContratConditions` en vous fondant sur la Figure 17.14. Commencez par placer les deux contrôles `Frame` principaux. À l'intérieur du premier (`Ouvrage`), placez deux contrôles `TextBox` et un contrôle `Label` au-dessus de chacun de ceux-ci afin d'en libeller la fonction (Titre de l'ouvrage, ISBN).

Placez deux autres contrôles `Frame` à l'intérieur du second cadre. Libellez le premier `Droit d'auteur` et placez à l'intérieur de celui-ci un contrôle `CheckBox`. Placez deux contrôles `TextBox` pour indiquer les taux des droits d'auteur et accolez-leur deux contrôles `SpinButton` qui permettront d'en modifier les valeurs.

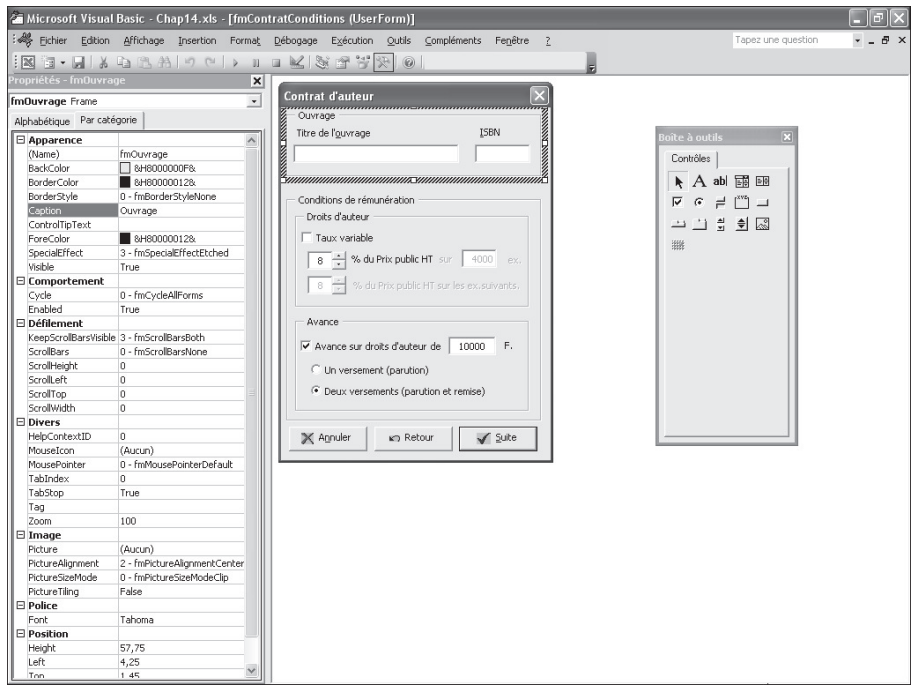
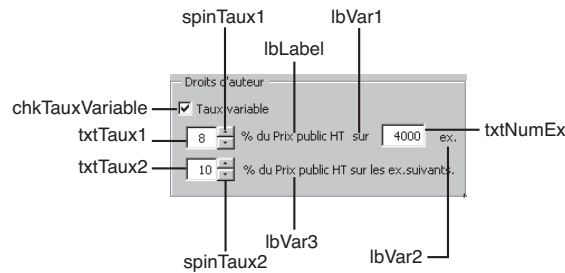


Figure 17.14
La feuille fmContratConditions en mode Conception

À droite du premier contrôle TextBox, placez deux contrôles Label suivis d'un autre contrôle TextBox et d'un dernier contrôle Label. Enfin, placez un contrôle Label à droite de la seconde zone de texte destinée à recevoir le taux des droits d'auteur. Appuyez-vous sur la Figure 17.15 et sur le tableau suivant pour définir les propriétés Name de ces contrôles.

Notez que nous avons séparé le libellé "% du prix HT" et "sur" de façon à pouvoir rendre inaccessible le libellé "sur", ainsi que tous les contrôles qui ne devront être renseignés que si le taux est variable.

Figure 17.15
Le cadre Droits d'auteur de la feuille fmContratAuteur.



Dans le cadre Avance, placez un contrôle CheckBox suivi d'un contrôle Label, d'un contrôle TextBox et d'un autre contrôle Label. Placez ensuite deux boutons d'option. Enfin, placez trois boutons de commande en dehors des cadres.

Affectez les propriétés suivantes aux contrôles :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmContratConditions
Caption	Contrat d'auteur
Contrôle Frame 1	
Caption	Ouvrage
Contrôle Frame 2	
Caption	Conditions de rémunération
Contrôle Frame 2.1	
Caption	Droits d'auteur
Contrôle Frame 2.2	
Caption	Avance
Contrôles du Frame 1	
Contrôle TextBox 1	
Name	txtTitre
Contrôle TextBox 2	
Name	txtISBN
Contrôle Label 1	
Caption	Titre de l'ouvrage
Contrôle Label 2	
Caption	ISBN
Contrôles du Frame 2.1	
Contrôle CheckBox	
Name	chkTauxVariable
Caption	Taux variable
Value	False

<i>Propriété</i>	<i>Valeur</i>
Contrôle TextBox 1	
Name	txtTaux1
Value	8
Contrôle TextBox 2	
Name	txtTaux2
Value	8
Contrôle TextBox 3	
Name	txtNumEx
Value	4000
Enabled	False
Contrôle SpinButton 1	
Name	spinTaux1
Value	8
Min	1
Max	20
Contrôle SpinButton 2	
Name	spinTaux2
Value	8
Min	1
Max	20
Enabled	False
Contrôle Label 1	
Name	Label1
Caption	% du prix public HT
Enabled	True
Contrôle Label 2	
Name	lbVar1
Caption	sur
Enabled	False

<i>Propriété</i>	<i>Valeur</i>
Contrôle Label 3	
Name	lbVar2
Caption	sur
Enabled	False
Contrôle Label 4	
Name	lbVar3
Caption	% du prix HT sur les ex. suivants
Enabled	False
Contrôles du Frame 2.2	
Contrôle CheckBox	
Name	chkAvance
Caption	Avances sur droit d'auteur
Value	True
Contrôle TextBox 1	
Name	txtAvance
Value	10000
Contrôle Label 1	
Name	lbAvance1
Caption	de
Contrôle Label 2	
Name	lbAvance2
Caption	F.
Contrôle OptionButton 1	
Name	optUnVersement
Caption	Un versement (parution)
Value	False
Contrôle OptionButton 2	
Name	optDeuxVersements
Caption	Deux versements (parution et remise)
Value	True

<i>Propriété</i>	<i>Valeur</i>
Boutons de commande	
Contrôle CommandButton 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle CommandButton 2	
Name	cmdRetour
Caption	Retour
Contrôle CommandButton 3	
Name	CmdSuite
Caption	Suite
Default	True

Placez ensuite le code suivant dans la fenêtre Code de la feuille :

```

1: Private Sub UserForm_Initialize()
2:     TxtTitre.SetFocus
3: End Sub
4:
5: Private Sub chkTauxVariable_Click()
6:     If chkTauxVariable.Value = True Then
7:         LbVar1.Enabled = True
8:         lbVar2.Enabled = True
9:         lbVar3.Enabled = True
10:        TxtTaux2.Enabled = True
11:        SpinTaux2.Enabled = True
12:        TxtNumEx.Enabled = True
13:    Else
14:        LbVar1.Enabled = False
15:        lbVar2.Enabled = False
16:        lbVar3.Enabled = False
17:        TxtTaux2.Enabled = False
18:        SpinTaux2.Enabled = False
19:        TxtNumEx.Enabled = False
20:    End If
21: End Sub
22:
23: Private Sub TxtTaux1_Change()

```



```
24: If IsNumeric(TxtTaux1.Value) = False Then
25:     MsgBox "Le taux pour les droits d'auteur doit être une valeur numérique.", _
26:     vbOKOnly + vbInformation, "Informations incomplètes"
27:     TxtTaux1.Value = ""
28:     TxtTaux1.SetFocus
29:     Exit Sub
30: ElseIf TxtTaux1.Value < 1 Or TxtTaux1.Value > 20 Then
31:     MsgBox "Le taux pour les droits d'auteur doit être compris entre 1 et 20.", _
32:     vbOKOnly + vbInformation, "Valeur non valide"
33:     TxtTaux1.Value = ""
34:     TxtTaux1.SetFocus
35: End If
36: spinTaux1.Value = TxtTaux1.Value
37: End Sub
38:
39: Private Sub TxtTaux2_Change()
40:     If IsNumeric(TxtTaux2.Value) = False Then
41:         MsgBox "Le taux pour les droits d'auteur doit être une valeur
42:         numérique.", _
43:         vbOKOnly + vbInformation, "Informations incomplètes"
44:         TxtTaux2.Value = ""
45:         TxtTaux2.SetFocus
46:         Exit Sub
47: ElseIf TxtTaux2.Value < 1 Or TxtTaux2.Value > 20 Then
48:         MsgBox "Le taux pour les droits d'auteur doit être compris entre 1 et 20.", _
49:         vbOKOnly + vbInformation, "Valeur non valide"
50:         TxtTaux2.Value = ""
51:         TxtTaux2.SetFocus
52:         Exit Sub
53: End If
54: SpinTaux2.Value = TxtTaux1.Value
55: End Sub
56: Private Sub spinTaux1_Change()
57:     TxtTaux1.Value = spinTaux1.Value
58: End Sub
59:
60: Private Sub SpinTaux2_Change()
61:     TxtTaux2.Value = SpinTaux2.Value
62: End Sub
63:
64: Private Sub chkAvance_Click()
65:     If chkAvance.Value = True Then
66:         lbAvance1.Visible = True
67:         lbAvance2.Visible = True
```

```
68:     txtAvance.Visible = True
69:     optUnVersement.Enabled = True
70:     OptDeuxVersements.Enabled = True
71: Else
72:     lbAvance1.Visible = False
73:     lbAvance2.Visible = False
74:     txtAvance.Visible = False
75:     optUnVersement.Enabled = False
76:     OptDeuxVersements.Enabled = False
77: End If
78: End Sub
79:
80: Private Sub cmdAnnuler_Click()
81:     Dim rep As Byte
82:     rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
cours ?", _
83: vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
84:     If rep = vbNo Then
85:         Exit Sub
86:     End If
87:     Me.Hide
88:     Call ContratFin
89:     End
90: End Sub
91:
92: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
93:     Dim rep As Byte
94:     rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en cours ?", _
95: vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
96:     If rep = vbNo Then
97:         Exit Sub
98:     End If
99:     Me.Hide
100:     Call ContratFin
101:     End
102: End Sub
103:
104: Private Sub CmdRetour_Click()
105:     fmContratAuteur.Show
106: End Sub
107:
108: Private Sub CmdSuite_Click()
109:
110: 'Vérifier validité des informations
```

```
111:     If TxtTitre.Value = "" Then
112:         MsgBox «Vous devez indiquer un titre d'ouvrage.», _
113:             vbOKOnly + vbInformation, «Informations incomplètes»
114:         TxtTitre.SetFocus
115:         Exit Sub
116:     End If
117:
118:     If TxtISBN.Value = "" Then
119:         MsgBox «Vous devez indiquer un ISBN.», _
120:             vbOKOnly + vbInformation, «Informations incomplètes»
121:         TxtISBN.SetFocus
122:         Exit Sub
123:     End If
124:     If Len(TxtISBN.Value) <> 6 Then
125:         MsgBox «L'ISBN n'est pas correctement formaté.», _
126:             vbOKOnly + vbInformation, «Informations incomplètes»
127:         TxtISBN.SetFocus
128:         Exit Sub
129:     End If
130:
131:     If TxtTaux1.Value = "" Then
132:         MsgBox «Vous devez indiquer un taux pour les droits d'auteur.», _
133:             vbOKOnly + vbInformation, «Informations incomplètes»
134:         TxtTaux1.SetFocus
135:         Exit Sub
136:     End If
137:
138:     If TxtTaux2.Value = "" And chkTauxVariable = True Then
139:         MsgBox «Vous devez indiquer un taux pour les droits d'auteur.», _
140:             vbOKOnly + vbInformation, «Informations incomplètes»
141:         TxtTaux2.SetFocus
142:         Exit Sub
143:     End If
144:
145:     If TxtNumEx.Value = "" And chkTauxVariable = True Then
146:         MsgBox «Vous devez indiquer une valeur valide pour le nbre
147:             d'exemplaires affectés par le taux 1.», _
148:             vbOKOnly + vbInformation, «Informations incomplètes»
149:         TxtNumEx.SetFocus
150:         Exit Sub
151:     End If
152:     If IsNumeric(TxtNumEx.Value) = False And chkTauxVariable = True Then
153:         MsgBox «Le nbre d'ex. concernés par le taux doit être une valeur numérique.», _
154:             vbOKOnly + vbInformation, «Informations incomplètes»
155:         TxtNumEx.Value = «»
```

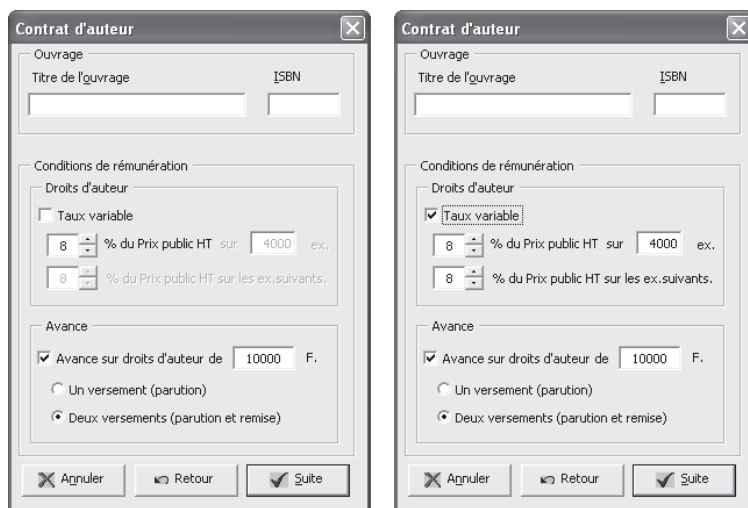
```
155:     TxtNumEx.SetFocus
156:     Exit Sub
157: End If
158:
159: If txtAvance.Value = "" And chkAvance = True Then
160:     MsgBox «Vous devez indiquer une somme pour l'avance sur droits d'auteur.», _
161:     vbOKOnly + vbInformation, «Informations incomplètes»
162:     txtAvance.SetFocus
163:     Exit Sub
164: End If
165: If IsNumeric(txtAvance.Value) = False And chkAvance = True Then
166:     MsgBox «L'avance sur droits d'auteur doit être une valeur numérique.», _
167:     vbOKOnly + vbInformation, «Informations incomplètes»
168:     txtAvance.Value = ""
169:     txtAvance.SetFocus
170:     Exit Sub
171: End If
172:
173: 'Validation des données
174: With MesConditionsAuteur
175:     .Titre = TxtTitre.Value
176:     .ISBN = TxtISBN.Value
177:     .TauxVariable = chkTauxVariable.Value
178:     .TauxDroitsNum1 = TxtTaux1.Value
179:     .TauxDroitsNum2 = TxtTaux2.Value
180:     .NumExemplairesTaux1 = TxtNumEx.Value
181:     .AvanceOuNon = chkAvance.Value
182:     If MesConditionsAuteur.AvanceOuNon = True Then
183:         MesConditionsAuteur.AvanceSurDroits = txtAvance.Value
184:     End If
185:     If optUnVersement = True Then
186:         MesConditionsAuteur.NumPaielements = 1
187:     Else
188:         MesConditionsAuteur.NumPaielements = 2
189:     End If
190:     'Obtention des équivalents chaînes des valeurs
191:     MesConditionsAuteur.TauxDroitsStr1 = RenvoyerChaîneValeur
192:     (.TauxDroitsNum1)
193:     MesConditionsAuteur.TauxDroitsStr2 = RenvoyerChaîneValeur
194:     (.TauxDroitsNum2)
195: End With
196: Me.Hide
197: End Sub
```

La procédure d'initialisation (lignes 1 à 3) a pour seule fonction de passer le focus au contrôle txtTitre.

La procédure chkTauxVariable_Click (lignes 5 à 21) gère l'événement clic survenant sur la case à cocher Taux variable. Si la case est cochée, les trois libellés txtVar1, txtVar2 et txtVar3 deviennent disponibles. Il en est de même pour les zones de texte txtTaux2 et le bouton toupie qui lui est attaché (spinTaux2), ainsi que pour la zone de texte txtNumEx (lignes 7 à 12). Si la case est décochée, tous ces contrôles sont désactivés (voir Figure 17.16).

Figure 17.16

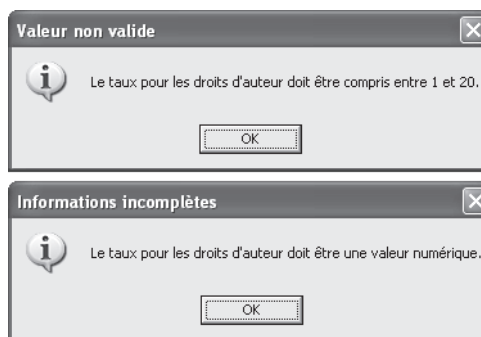
L'accessibilité des contrôles du cadre Droits d'auteur dépend de l'état de la case à cocher Taux variable.



Les procédures txtTaux1_Change et txtTaux2_Change (lignes 23 à 54) gèrent de façon identique les modifications apportées aux zones de texte destinées à recevoir les droits d'auteur. Lignes 24 à 29 et 40 à 45, on vérifie que la valeur entrée est une valeur numérique. Dans le cas contraire, un message s'affiche à l'attention de l'utilisateur, la valeur du contrôle réinitialisée et le focus lui est passé. Une instruction Exit entraîne la sortie de la procédure. Lignes 30 à 34 et 46 à 51, les mêmes instructions sont appliquées si la valeur saisie n'est pas comprise entre 1 et 20. Enfin, si la valeur entrée est valide, les instructions des lignes 36 et 53 affectent la valeur entrée par l'utilisateur dans l'une des zones de texte au bouton toupie associé. Cela est indispensable pour que, lorsque l'utilisateur clique sur le bouton toupie, la valeur affichée dans la zone de texte soit toujours en rapport avec la valeur de cette zone au moment du clic.

Figure 17.17

La validité des taux de droits d'auteur est contrôlée.

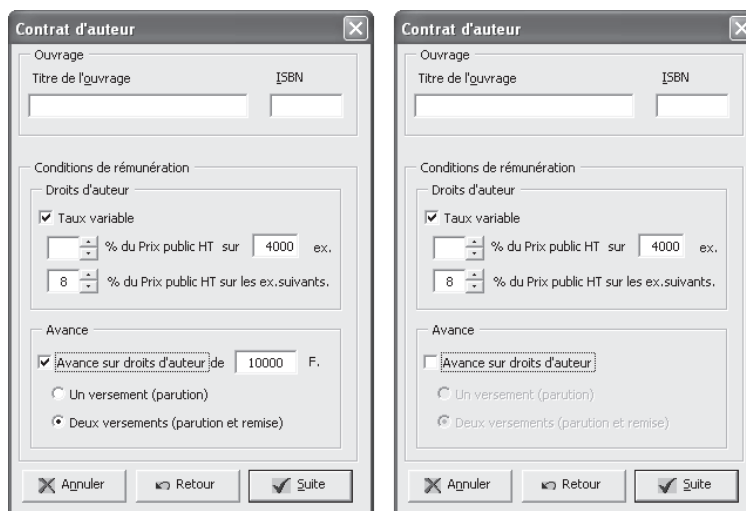


Les procédures des lignes 56 à 62 affectent la valeur des boutons toupie aux zones de texte associées lorsque l'utilisateur clique sur l'un d'eux.

La procédure `chkAvance_Click` (lignes 64 à 78) gère les clics sur la case à cocher `Avance` sur droits d'auteur. Si la case est cochée, les libellés `1bAvance1` et `1bAvance2` sont visibles (lignes 66 et 67). Il en est de même pour la zone de texte `txtAvance`. Enfin, les deux boutons d'option deviennent accessibles (lignes 69 et 70). Si la case est décochée, le traitement inverse est appliqué à ces contrôles (voir Figure 17.18).

Figure 17.18

L'état de la case à cocher `Avance` sur droits d'auteur influe sur les propriétés des autres contrôles du cadre `Avance`.



Lignes 80 à 102, les procédures `CmdAnnuler_Click` et `UserForm_QueryClose` gèrent les annulations. Elles sont semblables aux procédures de la feuille `fmContratAuteur` portant le même nom.

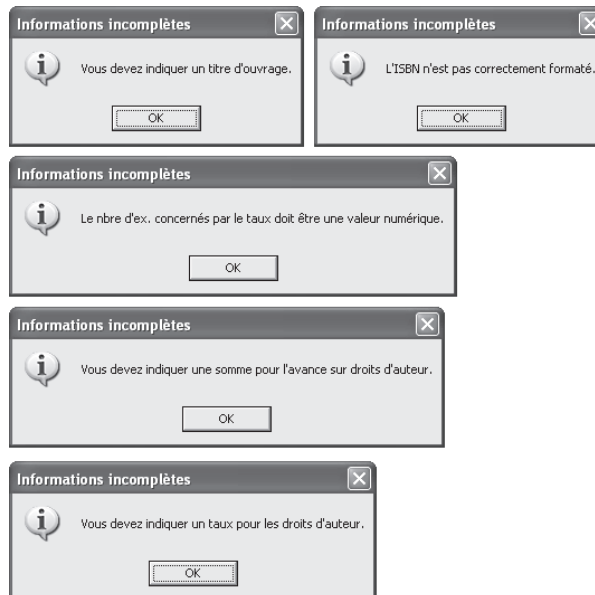
La procédure `CmdRetour_Click` des lignes 104 à 106 gère les clics sur le bouton Retour. Elle affiche la feuille `fmContratAuteur`. Notez que la feuille `fmContratConditions` n'est pas masquée. Si vous souhaitez la masquer, ajoutez simplement l'instruction `Me.Hide` devant l'instruction `fmContratAuteur.Show`.

Enfin, la procédure `CmdSuite_Click` gère les clics sur le bouton Suite. Lignes 110 à 171, la validité des données entrées est contrôlée. Le reste de la procédure valide les données en les affectant à la variable `MesConditionsAuteur`.

Lignes 111 à 123, on vérifie que les zones de texte Titre et ISBN contiennent des informations. Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur, le focus est passé au contrôle vide et une instruction `Exit` entraîne la sortie de la procédure. Ligne 124 à 129, on vérifie que l'ISBN entré contient six caractères. Lignes 131 à 136, on vérifie qu'un taux a été précisé pour les droits d'auteur dans le contrôle `txtTaux1`. L'instruction conditionnelle des lignes 138 à 143 vérifie que si la case Taux variable est cochée le second taux a bien été entré. Lignes 145 à 150, on contrôle que si la case Taux variable est cochée, le nombre d'exemplaires touchés par le taux initial a été fourni. On vérifie ensuite si ce taux est une valeur numérique (lignes 151 à 157). Enfin, lignes 159 à 170, on vérifie que si la case Avance est cochée, une valeur a été fournie (lignes 159 à 164) et que cette valeur est une valeur numérique (lignes 165 à 171).

Figure 17.19

La validité des données est contrôlée.



Si les données fournies sont valides, la procédure atteint la ligne 173 sans qu'une instruction `Exit` n'en ait entraîné la sortie. Les données sont alors affectées aux différents sous-types

de la variable `MesConditionsAuteur` à l'aide d'une structure `With...End With`. Remarquez l'instruction conditionnelle utilisée lignes 185 à 189 pour affecter la valeur 1 ou 2 à `MesConditionsAuteur`, selon que l'un ou l'autre des boutons d'option est coché.

Lignes 191 et 192, la fonction `RenvoyerChaîne` est appelée à deux reprises et reçoit chaque fois une valeur correspondant à l'un des taux de droits d'auteur sous forme numérique. Cette fonction doit renvoyer la valeur qui lui est passée sous forme de chaîne de caractères, pour les besoins du contrat. Placez la fonction `RenvoyerChaîne` dans le module `Contrats-Divers`. Elle se présente comme suit :

```
Public Function RenvoyerChaîneValeur(valeur)
    Select Case valeur
        Case 0
            RenvoyerChaîneValeur = "zéro"
        Case 1
            RenvoyerChaîneValeur = "un"
        Case 2
            RenvoyerChaîneValeur = "deux"
        Case 3
            RenvoyerChaîneValeur = "trois"
        Case 4
            RenvoyerChaîneValeur = "quatre"
        Case 5
            RenvoyerChaîneValeur = "cinq"
        Case 6
            RenvoyerChaîneValeur = "six"
        Case 7
            RenvoyerChaîneValeur = "sept"
        Case 8
            RenvoyerChaîneValeur = "huit"
        Case 9
            RenvoyerChaîneValeur = "neuf"
        Case 10
            RenvoyerChaîneValeur = "dix"
        Case 11
            RenvoyerChaîneValeur = "onze"
        Case 12
            RenvoyerChaîneValeur = "douze"
        Case 13
            RenvoyerChaîneValeur = "treize"
        Case 14
            RenvoyerChaîneValeur = "quatorze"
        Case 15
            RenvoyerChaîneValeur = «quinze»
```



```

Case 16
    RenvoyerChaineValeur = «seize»
Case 17
    RenvoyerChaineValeur = «dix-sept»
Case 18
    RenvoyerChaineValeur = «dix-huit»
Case 19
    RenvoyerChaineValeur = «dix-neuf»
Case 20
    RenvoyerChaineValeur = «vingt»
End Select
End Function

```

Cette fonction utilise une structure `Select Case...End Select` pour renvoyer une chaîne en fonction de la valeur qui lui est passée.



À ce stade, placez l'instruction d'affichage de la feuille `fmContratConditions` dans la procédure `EditionContratAuteur` et exécutez cette dernière afin de vérifier que votre programme ne contient pas d'erreurs. Testez les différentes conditions en entrant des données variables dans les interfaces. La procédure `EditionContratAuteur` doit maintenant se présenter comme suit :

```

Sub EditionContratAuteur()
    Set ClasseurAuteurs = GetObject(«C:\Mes documents\Classeur
Auteurs.xlsx»)
    Set MonTableauWord = GetObject(, «Word.Application»)
    If Err.Number <> 0 Then Err.Clear
    fmContratAuteur.Show
    fmContratConditions.Show
End Sub

```

Feuille *fmContratDates*

La feuille `fmContratDates` permettra à l'utilisateur d'indiquer les dates de remise et de parution. Nous utiliserons pour cela le contrôle `Calendar` sur lequel l'utilisateur n'aura qu'à cliquer pour préciser une date. Une zone de liste modifiable permettra de sélectionner la date que l'utilisateur veut préciser (date de remise ou date de parution). Deux zones de texte dont la fonction sera précisée par deux contrôles `Label` laisseront apparaître les dates sélectionnées sur le calendrier. Ces zones de texte permettront de visualiser les dates choisies, mais leur valeur ne pourra être modifiée. Enfin, nous retrouverons les mêmes boutons de commande que pour la feuille `fmContratConditions`, à savoir un bouton `Annuler`, un bouton `Retour` et un bouton `Suite`. La Figure 17.20 représente la feuille `fmContratDates` en mode Conception.

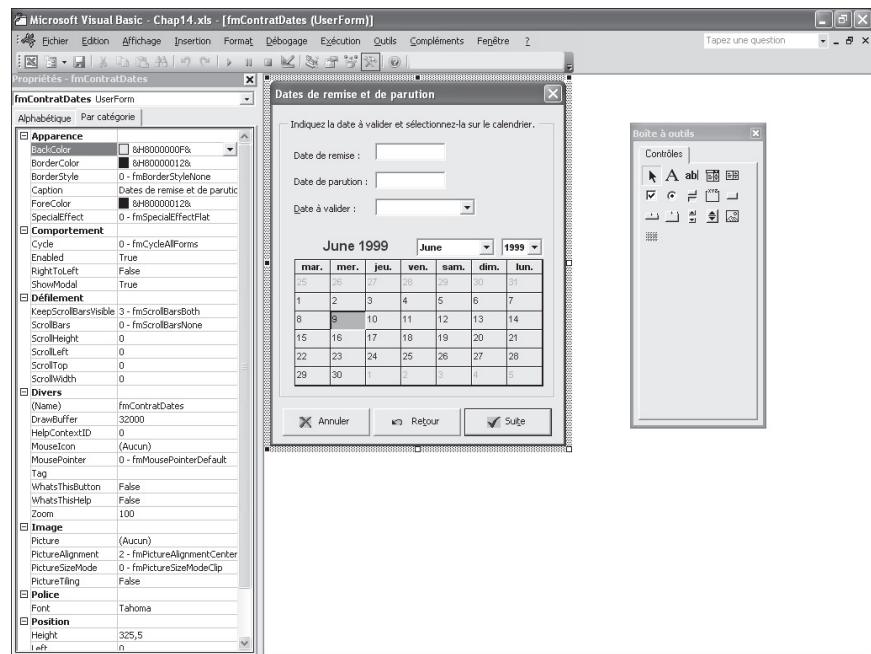


Figure 17.20
La feuille fmContratDates en mode Conception.



Le contrôle Calendar ne fait pas partie des contrôles standard de la boîte à outils de Visual Basic Editor. Pour un rappel de la procédure permettant d'ajouter un contrôle à la boîte à outils, consultez la section "Personnaliser la boîte à outils" du Chapitre 12.

Pour "dessiner" la feuille fmContratDates, commencez par placer un contrôle Frame sur la feuille. Placez ensuite deux contrôles TextBox et un contrôle ComboBox dans le cadre. Déposez trois contrôles Label à côté de chacun d'eux afin d'en préciser la fonction. Déposez et dimensionnez le contrôle Calendar. Déposez à l'extérieur du cadre les trois boutons de commande.



Copiez les boutons de la feuille fmContratConditions et collez-les sur la feuille fmContratDates. Ainsi vous n'aurez pas à en redéfinir les propriétés. Vous pouvez également copier-coller les procédures cmdAnnuler_Click et UserForm_QueryClose puisque celles-ci sont identiques d'une feuille à l'autre.

Définissez les propriétés des contrôles comme indiqué dans le tableau suivant :

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmContratDates
Caption	Dates de remise et de parution
Contrôle Frame	
Caption	Indiquez la date à valider et sélectionnez-la sur le calendrier
Contrôle TextBox 1	
Name	txtRemise
Enabled	False
Contrôle TextBox 2	
Name	txtParution
Enabled	False
Contrôle ComboBox	
Name	cboDateAValider
Style	2 - fmStyleDropDownList
Contrôle Label 1	
Caption	Date de remise :
Contrôle Label 2	
Caption	Date de parution :
Contrôle Label 3	
Caption	Date à valider :
Contrôle Calendar	
Name	Calendrier
Contrôle CommandButton 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle CommandButton 2	
Name	cmdRetour
Caption	Retour
Contrôle CommandButton 3	
Name	CmdSuite
Caption	Suite
Default	True

Ouvrez ensuite la fenêtre Code de la feuille et placez-y les procédures événementielles reproduites ci-après :

```
1: Private Sub UserForm_Initialize()
2:   cboDateAValider.AddItem ("Date de remise")
3:   cboDateAValider.AddItem ("Date de parution")
4:   cboDateAValider.ListIndex = 0
5:   Calendrier.Value = Date + 30
6:   Calendrier.SetFocus
7: End Sub
8:
9: Private Sub Calendrier_Click()
10:  If cboDateAValider.ListIndex = 0 Then
11:    txtRemise.Value = Calendrier.Value
12:    cboDateAValider.ListIndex = 1
13:  Else
14:    TxtParution.Value = Calendrier.Value
15:  End If
16: End Sub
17:
18: Private Sub CmdAnnuler_Click()
19:  Dim rep As Byte
20:  rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
    cours ?", _
21:  vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
22:  If rep = vbNo Then
23:    Exit Sub
24:  End If
25:  Me.Hide
26:  Call ContratFin
27:  End
28: End Sub
29:
30: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
31:  Dim rep As Byte
32:  rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en
    cours ?", _
33:  vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
34:  If rep = vbNo Then
35:    Exit Sub
36:  End If
37:  Me.Hide
38:  Call ContratFin
39:  End
```

```
40: End Sub
41:
42: Private Sub CmdRetour_Click()
43: fmContratConditions.Show
44: End Sub
45:
46: Private Sub CmdSuite_Click()
47:
48: 'Vérifier la validité des informations
49:   If txtRemise.Value = "" Then
50:     MsgBox «Vous devez spécifier une date de remise.», _
51:     vbOKOnly + vbCritical, «Contrat d'auteur»
52:     Exit Sub
53:   ElseIf TxtParution.Value = "" Then
54:     MsgBox «Vous devez spécifier une date de parution.», _
55:     vbOKOnly + vbCritical, «Contrat d'auteur»
56:     Exit Sub
57:   ElseIf DateValue(TxtParution) - DateValue(txtRemise) <= 0 Then
58:     MsgBox «Vous devez spécifier une date de remise antérieure à la date de
59:     parution.», _
60:     vbOKOnly + vbCritical, «Contrat d'auteur»
61:     Exit Sub
62:   ElseIf DateValue(TxtParution) - DateValue(txtRemise) < 40 Then
63:     Dim continuer As Integer
64:     continuer = MsgBox(«Vous avez indiqué une date de remise à seulement « _
65:     & DateValue(TxtParution) - DateValue(txtRemise) _
66:     & « jours de la date de parution. Valider cette date ?»,
67:     vbYesNo + vbQuestion, _
68:     «Valider la date de remise ?»)
69:     If continuer = vbNo Then Exit Sub
70:   End If
71:
72:   Me.Hide
73:
74: 'Validation des dates
75:   MesConditionsAuteur.remise = txtRemise.Value
76:   MesConditionsAuteur.Parution = TxtParution.Value
77: End Sub
```

La procédure d'initialisation (lignes 1 à 7) ajoute les entrées Date de remise et Date de parution à la zone de liste modifiable (lignes 2 et 3), puis active le premier élément de la liste (ligne 4). Ligne 5 et 6, la valeur du calendrier est définie à 30 jours de la date courante – renvoyée par la fonction Date –, et le calendrier reçoit le focus.

La procédure `Calendrier_Click` (lignes 9 à 16) gère l'affectation des dates sélectionnées sur le calendrier aux zones de texte `txtRemise` et `txtParution`. Une structure conditionnelle est utilisée pour définir l'entrée sélectionnée dans la zone de liste modifiable et entrer la date choisie dans la zone de texte correspondante (lignes 11 et 14). Si le premier élément de la liste est sélectionné (Date de remise), la zone de texte `txtRemise` reçoit pour valeur la date sélectionnée et le second élément (Date de parution) est sélectionné dans la zone de liste modifiable (ligne 12). Ainsi, l'utilisateur n'a pas à modifier l'entrée sélectionnée dans la zone de liste modifiable et peut choisir immédiatement la date de parution.

Les procédures `CmdAnnuler_Click` et `UserForm_QueryClose` (lignes 18 à 40) sont identiques aux procédures équivalentes des feuilles `fmContratAuteur` et `fmContratConditions`.

La procédure `CmdRetour_Click` (lignes 42 à 44) gère les clics sur le bouton Retour. Elle affiche la feuille `fmContratConditions`.

Enfin, la procédure `CmdSuite_Click` (lignes 46 à 75) vérifie la validité des dates indiquées (lignes 48 à 68) et affecte ces valeurs aux variables appropriées.

Une seule instruction conditionnelle est utilisée pour vérifier la validité des dates. Lignes 49 à 56, on vérifie qu'une date de remise et une date de parution ont été précisées. Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur et une instruction `Exit` entraîne la sortie de la procédure (voir Figure 17.21).

Figure 17.21

Les dates de remise et de parution doivent être précisées.

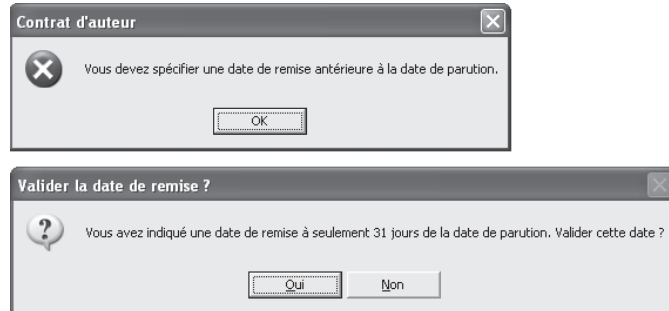


Lignes 57 à 60, on vérifie que la date de parution est postérieure à la date de remise. On utilise pour cela la fonction `DateValue` qui renvoie la valeur de la date sous forme numérique (consultez l'aide en ligne pour plus de précisions). Si tel n'est pas le cas, un message s'affiche à l'attention de l'utilisateur et une instruction `Exit` entraîne la sortie de la procédure. Lignes 61 à 68, on vérifie si plus de 40 jours séparent la date de remise de la date de parution. Si tel n'est pas le cas, l'utilisateur en est averti par un message. Il peut alors

valider les dates ou cliquer sur le bouton Non. Dans ce second cas de figure, une instruction Exit (ligne 67) entraîne la sortie de la procédure.

Figure 17.22

Les dates proposées sont comparées à l'aide de la fonction DateValue.



Ligne 70, la feuille est masquée. Les dates fournies sont ensuite affectées aux espaces de stockage appropriés de la variable MesConditionsAuteur.



Placez l'instruction d'affichage de la feuille fmContratDates dans la procédure EditionContratAuteur et exécutez cette dernière afin de vérifier que votre programme ne contient pas d'erreurs. La procédure EditionContratAuteur doit maintenant se présenter comme suit :

```
Sub EditionContratAuteur()
    Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur
Auteurs.xlsx")
    Set MonTableauWord = GetObject(, "Word.Application")
    If Err.Number <> 0 Then Err.Clear
    fmContratAuteur.Show
    fmContratConditions.Show
    fmContratDates.Show
End Sub
```

Feuille *fmContratImpression*

La feuille fmContratImpression permettra à l'utilisateur d'indiquer s'il souhaite imprimer le contrat et les feuilles de paie, ainsi que le nombre de copies à réaliser. Une case à cocher servira à préciser s'il souhaite ou non imprimer. Deux autres cases à cocher permettront d'indiquer si les feuilles de paie, le contrat ou les deux doivent être imprimés. Elles ne seront accessibles que si la case Imprimer est cochée. Une zone de texte et un bouton toupie permettront de préciser le nombre de copies souhaitées pour chaque impression. Enfin, nous retrouverons les trois boutons de retour, d'annulation et de validation des feuilles

précédemment créées. La Figure 17.23 présente la feuille `fmContratImpression` en mode Conception.

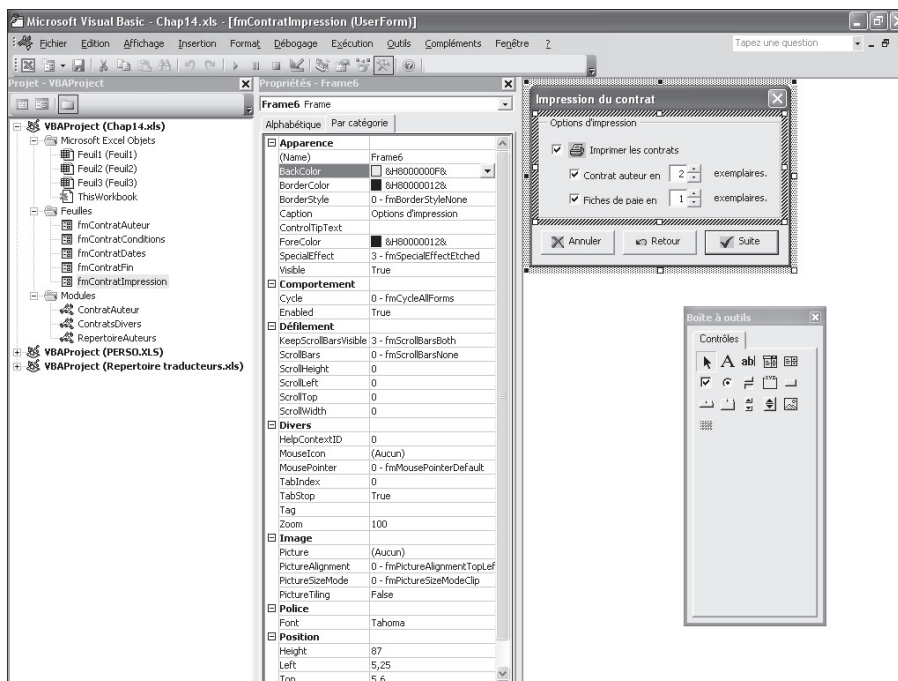


Figure 17.23
La feuille fmContratImpression en mode Conception.



Notez l'image de l'imprimante apparaissant entre la première case à cocher et son libellé. Il s'agit d'un contrôle Picture qui a été placé sur le contrôle Check-Box. Nous avons ajouté des espaces devant le texte de la propriété Caption du contrôle afin de ne pas masquer le libellé de la case à cocher.

Commencez par placer un contrôle Frame sur la feuille. Placez ensuite trois cases à cocher sur la feuille, en plaçant les deux dernières en retrait par rapport à la première. Déposez un contrôle TextBox, un contrôle SpinButton et un contrôle Label à côté de ces deux cases à cocher. Enfin, placez trois boutons de commande hors du cadre. Définissez les propriétés des contrôles comme indiqué dans le tableau suivant.

<i>Propriété</i>	<i>Valeur</i>
Feuille	
Name	fmContratImpression
Caption	Impression du contrat
Contrôle Frame	
Caption	Options d'impression
Contrôle CheckBox 1	
Name	chk Imprimer
Caption	Imprimer les contrats
Value	True
Contrôle CheckBox 2	
Name	chkImprimessionContrat
Caption	Contrat auteur en
Value	True
Contrôle CheckBox 3	
Name	chkImprimessionPaie
Caption	Fiches de paie en
Value	True
Contrôle TextBox 1	
Name	txtQteContrat
Value	2
Contrôle TextBox 2	
Name	txtQtePaie
Value	1
Contrôle SpinButton 1	
Name	spinContrat
Value	2
Min	1
Max	10

<i>Propriété</i>	<i>Valeur</i>
Contrôle SpinButton 2	
Name	spinPaie
Value	1
Min	1
Max	10
Contrôles Label 1 et 2	
Caption	Exemplaires
Contrôle CommandButton 1	
Name	cmdAnnuler
Caption	Annuler
Cancel	True
Contrôle CommandButton 2	
Name	cmdRetour
Caption	Retour
Contrôle CommandButton 3	
Name	CmdTerminer
Caption	Terminer
Default	True

Ouvrez la fenêtre Code de la feuille et placez-y le code des procédures événementielles que devra gérer la feuille :

```

1: Private Sub UserForm_Initialize()
2:     If MonAuteur.societe = True Then
3:         ChkImpressionPaie.Value = False
4:         ChkImpressionPaie.Enabled = False
5:         TxtQtePaie.Enabled = False
6:         SpinPaie.Enabled = False
7:     End If
8: End Sub
9:
10: Private Sub ChkImprimer_Click()
11:     If ChkImprimer.Value = False Then
12:         ChkImpressionPaie.Enabled = False

```

```
13:      ChkImpressionContrat.Enabled = False
14:      TxtQteContrat.Locked = True
15:      TxtQtePaie.Locked = True
16:      SpinPaie.Enabled = False
17:      spinContrat.Enabled = False
18:  Else
19:      If MonAuteur.societe = False Then
20:          ChkImpressionPaie.Enabled = True
21:          TxtQtePaie.Locked = False
22:          SpinPaie.Enabled = True
23:      End If
24:      ChkImpressionPaie.Value = True
25:      ChkImpressionContrat.Enabled = True
26:      ChkImpressionContrat.Value = True
27:      TxtQteContrat.Locked = False
28:      spinContrat.Enabled = True
29:  End If
30: End Sub
31:
32: Private Sub chkImpressionContrat_Click()
33:     If ChkImpressionContrat.Value = False Then
34:         TxtQteContrat.Locked = True
35:         spinContrat.Enabled = False
36:     Else
37:         TxtQteContrat.Locked = False
38:         spinContrat.Enabled = True
39:     End If
40: End Sub
41:
42: Private Sub ChkImpressionPaie_Click()
43:     If ChkImpressionPaie.Value = False Then
44:         TxtQtePaie.Locked = True
45:         SpinPaie.Enabled = False
46:     Else
47:         TxtQtePaie.Locked = False
48:         SpinPaie.Enabled = True
49:     End If
50: End Sub
51:
52: Private Sub spinContrat_Change()
53:     TxtQteContrat.Value = spinContrat.Value
54: End Sub
55:
```

```
56: Private Sub SpinPaie_Change()
57:     TxtQtePaie.Value = SpinPaie.Value
58: End Sub
59:
60: Private Sub TxtQteContrat_Change()
61:     If IsNumeric(TxtQteContrat.Value) = False Then
62:         MsgBox "Le nombre de copies doit être une valeur numérique.", _
63:             vbOKOnly + vbInformation, "Informations erronées"
64:         TxtQteContrat.Value = ""
65:         TxtQteContrat.SetFocus
66:         Exit Sub
67:     ElseIf TxtQteContrat.Value < 1 Or TxtQteContrat.Value > 10 Then
68:         MsgBox "Le nombre de copies doit être compris entre 1 et 10.", _
69:             vbOKOnly + vbInformation, "Valeur non valide"
70:         TxtQteContrat.Value = «»
71:         TxtQteContrat.SetFocus
72:         Exit Sub
73:     End If
74:     spinContrat.Value = TxtQteContrat.Value
75: End Sub
76:
77: Private Sub TxtQtePaie_Change()
78:     If IsNumeric(TxtQtePaie.Value) = False Then
79:         MsgBox "Le nombre de copies doit être une valeur numérique.", _
80:             vbOKOnly + vbInformation, "Informations erronées"
81:         TxtQtePaie.Value = ""
82:         TxtQtePaie.SetFocus
83:         Exit Sub
84:     ElseIf TxtQtePaie.Value < 1 Or TxtQtePaie.Value > 10 Then
85:         MsgBox "Le nombre de copies doit être compris entre 1 et 10.", _
86:             vbOKOnly + vbInformation, "Valeur non valide"
87:         TxtQtePaie.Value = ""
88:         TxtQtePaie.SetFocus
89:         Exit Sub
90:     End If
91:     SpinPaie.Value = TxtQtePaie.Value
92: End Sub
93:
94: Private Sub CmdAnnuler_Click()
95:     Dim rep As Byte
96:     rep = MsgBox("Êtes-vous sûr de vouloir annuler l'édition du contrat en
    ➡ cours ?", _
```

```
97:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
98:     If rep = vbNo Then Exit Sub
99:     Me.Hide
100: End
101: End Sub
102:
103: Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
104:     Dim rep As Byte
105:     rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en  

    ➡ cours ?", _
106:     vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")
107:     If rep = vbNo Then
108:         Exit Sub
109:     End If
110:     Me.Hide
111: End
112: End Sub
113:
114: Private Sub CmdRetour_Click()
115:     fmContratConditions.Show
116: End Sub
117:
118: Private Sub CmdSuite_Click()
119:     With MonImpression
120:         .Imprimer = ChkImprimer.Value
121:         .ImprimerCourrier = ChkImpressionContrat.Value
122:         .ImprimerPaie = ChkImpressionPaie.Value
123:         .nbreCourrier = TxtQteContrat.Value
124:         .nbrePaie = TxtQtePaie.Value
125:     End With
126:     Me.Hide
127: End Sub
```

La procédure d'initialisation de la feuille (lignes 1 à 8) utilise une structure conditionnelle pour désactiver les contrôles dédiés à l'impression des feuilles de paie si le contrat est édité au nom d'une société – les sociétés facturent tandis qu'on établit une feuille de paie au nom des personnes physiques. On interroge pour cela la valeur de la variable `MonAuteur`. `Societe` qui a reçu la valeur `True` ou `False` lorsque l'utilisateur a validé la feuille `fmContratAuteur`. Si la variable renvoie la valeur `True`, la case à cocher `chkImpressionPaie` est décochée (ligne 3), puis rendue inaccessible (ligne 4). La zone de texte et le bouton toupie associés sont également rendus inaccessibles (lignes 5 et 6).

La procédure `ChkImprimer_Click` modifie l'état des contrôles permettant de définir les éléments à imprimer et le nombre de copies à réaliser. Si le bouton est décoché, les deux autres cases à cocher sont désactivées (lignes 12 et 13), les deux zones de texte correspondantes sont verrouillées (lignes 14 et 15) et les deux boutons toupie associés à ces dernières rendus inaccessibles (lignes 16 et 17). Dans le cas contraire, les instructions des lignes 19 à 29 s'exécutent. Lignes 19 à 24, une structure conditionnelle imbriquée active les contrôles permettant de définir le nombre d'impression des feuilles de paie, si le contrat n'est pas établi au nom d'une société. Les contrôles servant à définir l'impression du contrat sont activés (lignes 25 à 28).

Lignes 32 à 40, la procédure `chkImpressionContrat_Click` gère les clics sur la case correspondante. Elle active ou désactive l'accès à la zone de texte et au bouton toupie correspondant, selon que la case est cochée ou décochée. Lignes 42 à 50, la procédure `ChkImpressionPaie_Click` effectue les mêmes opérations pour la case à cocher libellée *Fiches de paie*.

Les deux procédures des lignes 52 à 58 affectent la valeur du bouton toupie à la case correspondante lorsque l'utilisateur clique sur l'un des contrôles `SpinContrat` ou `spinPaie`.

Lignes 60 à 75 et lignes 77 à 92, les procédures `TxtQteContrat_Change` et `TxtQtePaie_Change` sont identiques aux procédures `TxtTaux1_Change` et `TxtTaux2_Change` de la feuille `fmContratConditions`. Elles permettent de s'assurer que les valeurs saisies dans les zones de texte sont valides et d'affecter la valeur saisie au contrôle `SpinButton` correspondant.

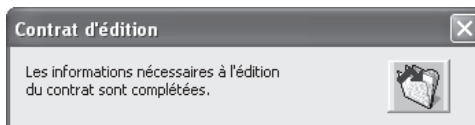
Les procédures `CmdAnnuler_Click` et `UserForm_QueryClose` sont identiques aux procédures équivalentes des feuilles `fmContratAuteur`, `fmContratConditions` et `fmContratDates`. La procédure `CmdRetour_Click` des lignes 114 à 116 affiche la feuille `fmContratDates`.

Enfin la procédure `CmdSuite_Click` des lignes 118 à 127 transfère les informations d'impression fournies dans la variable `MonImpression`.

Feuille *fmContratFin*

La feuille `fmContratFin` est très simple. Elle sert à indiquer à l'utilisateur que les informations nécessaires ont été fournies et que le contrat va être édité. Elle contient un seul bouton et un contrôle `Label`. Réalisez cette feuille en vous appuyant sur sa représentation de la Figure 17.24 (nous avons affecté une image bitmap à la propriété `Picture` du contrôle). Définissez la propriété `Caption` du bouton de commande à `cmdOK`.

Figure 17.24
Ouf !



Placez maintenant le code suivant dans la fenêtre Code de la feuille fmContratFin :

```
Private Sub cmdOK_Click()  
    Me.Hide  
End Sub  
  
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)  
    Dim rep As Byte  
    rep = MsgBox("Etes-vous sûr de vouloir annuler l'édition du contrat en  
cours ?", _  
        vbYesNo + vbQuestion, "Annuler l'édition de contrat ?")  
    If rep = vbNo Then  
        Exit Sub  
    End If  
    Me.Hide  
End Sub
```

On ne présente plus la procédure UserForm_QueryClose. Quant à la procédure CmdOK_Click, elle masque simplement la feuille.

À ce stade du développement, la procédure EditionContratAuteur doit se présenter comme suit :

```
Sub EditionContratAuteur()  
    '1. Liaisons des var. objet  
    Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")  
    Set MonTableauWord = GetObject(, "Word.Application")  
    If Err.Number <> 0 Then Err.Clear  
  
    '2. Affichage des feuilles  
    fmContratAuteur.Show  
    fmContratConditions.Show  
    fmContratDates.Show  
    fmContratImpression.Show  
    fmContratFin.Show  
    '3. Edition des feuilles de paie et du courrier  
    '4. Mise à jour du tableau Word  
    '5. libération des ressources mémoire  
End Sub
```

Écrire des procédures d'édition de documents

Les interfaces sont maintenant développées, les données stockées dans des variables. Il ne nous reste qu'à écrire les procédures qui généreront le contrat et les feuilles de paie, les imprimeront et les enregistreront, et qui mettront à jour le tableau Word.

Édition des feuilles de paie

Commencez par modifier la procédure EditionContratAuteur en y insérant l'appel de la procédure EditerFeuilleDePaie.

```
Sub EditionContratAuteur()

    '1. Liaisons des var. objet
    Set ClasseurAuteurs = GetObject(«C:\Mes documents\Classeur Auteurs.xlsx»)
    Set MonTableauWord = GetObject(, «Word.Application»)
    If Err.Number <> 0 Then Err.Clear

    '2. Affichage des feuilles
    fmContratAuteur.Show
    fmContratConditions.Show
    fmContratDates.Show
    fmContratImpression.Show
    fmContratFin.Show

    '3. Edition des feuilles de paie et du courrier
    If MonAuteur.societe = False Then
        Call EditerFeuilleDePaie
    End If

    '4. Mise à jour du tableau Word

    '5. libération des ressources mémoire

End Sub
```

Notez que la procédure EditerFeuilleDePaie n'est appelée que si le contrat n'est pas édité au nom d'une société.

Placez maintenant la procédure EditerFeuilleDePaie dans le module ContratAuteur :

```
1: Sub EditerFeuilleDePaie()
2:     Dim feuilnum As Byte
3:
4:     For feuilnum = 1 To MesConditionsAuteur.NumPaiements
5:         'Edition de la (des) feuille(s)
6:         Workbooks.Add Template:=»C:\Documents and settings\Administrateur\
           Application Data\Microsoft\Modèles\PAIEMENT.xlt"
7:         With ActiveWorkbook.Sheets(1)
8:             If MonAuteur.societe = False Then
9:                 .Range("C2").Value = MonAuteur.Prenom & " " & MonAuteur.Nom
10:            Else
```



```
11:         .Range("C2").Value = MonAuteur.Nom
12:     End If
13:
14:     .Range("C3").Value = MesConditionsAuteur.Titre
15:     .Range("C4").Value = MesConditionsAuteur.ISBN
16:     .Range("C6").Value = MesConditionsAuteur.remise
17:     .Range("C7").Value = MesConditionsAuteur.Parution
18:     .Range("C9").Value = MonAuteur.Adresse
19:     .Range("C10").Value = MonAuteur.CodePostal
20:     .Range("C11").Value = MonAuteur.Ville
21:     .Range("C12").Value = MonAuteur.Pays
22:     .Range("F15").Value = _
23:     MesConditionsAuteur.AvanceSurDroits / MesConditionsAuteur.NumPaiements
24:
25:     If MesConditionsAuteur.NumPaiements = 1 Then
26:         .Range("F30").Value = MesConditionsAuteur.Parution
27:     Else
28:         If feuillnum = 1 Then
29:             .Range("F30").Value = MesConditionsAuteur.remise
30:         Else
31:             .Range("F30").Value = MesConditionsAuteur.Parution
32:         End If
33:     End If
34:     If MesConditionsAuteur.TauxVariable = True Then
35:         .Range("B32") = «Pour les « & _
36:         MesConditionsAuteur.NumExemplairesTaux1 & _
37:         « premiers ouvrages»
38:         .Range("B33") = «Au-delà»
39:         .Range("F32") = MesConditionsAuteur.TauxDroitsNum1 & « %»
40:         .Range("G32") = MesConditionsAuteur.TauxDroitsStr1 & « pour cent»
41:         .Range("F33") = MesConditionsAuteur.TauxDroitsNum2 & « %»
42:         .Range("G33") = MesConditionsAuteur.TauxDroitsStr2 & « pour cent»
43:     Else
44:         .Range("B32") = " "
45:         .Range("B33") = «»
46:         .Range("F32") = MesConditionsAuteur.TauxDroitsNum1 & « %»
47:         .Range("G32") = MesConditionsAuteur.TauxDroitsStr1 & « pour cent»
48:         .Range("F33") = " "
49:         .Range("G33") = " "
50:     End If
51: End With
52: Application.Calculate
53:
```

```

54:      'impression du classeur
55:      If MonImpression.Imprimer = True And MonImpression.ImprimerPaie =
      ➡ True Then
56:          ActiveWorkbook.PrintOut copies:=MonImpression.nbrePaie
57:      End If
58:
59:      'enregistrement du classeur
60:      ActiveWorkbook.SaveAs Filename:="C:\Mes Documents\" & _
61:      MesConditionsAuteur.ISBN & " " & MonAuteur.Nom & " " & _
62:      «Paie» & feuilnum & ".xlsx", FileFormat:=xlNormal
63:      ActiveWorkbook.Close
64:      Next feuilnum
65: End Sub

```

Cette procédure utilise une boucle For...Next pour éditer le nombre de feuilles de paie approprié, soit `MesConditionsAuteur.NumPaiements`.

À chaque passage de la boucle, un classeur fondé sur le modèle `Paiement.xlt` est créé (ligne 6). Notez que le chemin apparaissant dans ce listing est celui du dossier de stockage des modèles sur la machine sur laquelle a été créé et testé le programme. Il se peut qu'il diffère sur votre machine. Lignes 7 à 34 une structure `With...End With` est utilisée pour insérer les données appropriées dans les cellules du classeur. Ligne 22, l'avance sur droits d'auteur est divisée par le nombre de paiements. Lignes 25 à 33, une structure conditionnelle sert à définir les dates de règlement : si un seul paiement est effectué, le règlement s'opère à la parution (ligne 26). Si le règlement est effectué en deux versements (lignes 27 à 33), le premier règlement est versé lors de la remise (ligne 29) et le second lors de la parution (ligne 31). Lignes 35 à 50, une structure conditionnelle sert à compléter les informations concernant les taux des droits d'auteur. Ligne 52, le calcul est forcé sur les calculs ouverts. Lignes 55 à 57 le classeur est imprimé. Les lignes 60 à 62 correspondent à une seule instruction d'enregistrement. Le classeur est enregistré dans le dossier `C:\Mes documents` et a pour nom *ISBN Nom Paie1.xlsx* ou *ISBN Nom Paie2.xlsx* – où *ISBN* est l'ISBN de l'ouvrage, et *Nom* le nom de l'auteur. Ligne 63, le classeur est fermé.

Mise à jour du Tableau Word

Commencez par insérer l'appel de la procédure de mise à jour du tableau dans la procédure `EditionContratAuteur`.

```

Sub EditionContratAuteur()

    '1. Liaisons des var. objet
    Set ClasseurAuteurs = GetObject("C:\Mes documents\Classeur Auteurs.xlsx")
    Set MonTableauWord = GetObject(, "Word.Application")

```

```
If Err.Number <> 0 Then Err.Clear

'2. Affichage des feuilles
fmContratAuteur.Show
fmContratConditions.Show
fmContratDates.Show
fmContratImpression.Show
fmContratFin.Show

'3. Edition des feuilles de paie et du courrier
If MonAuteur.societe = False Then
    Call EditerFeuilleDePaie
End If
Call MettreTableauAJour
'4. Mise à jour du tableau Word

'5. libération des ressources mémoire

End Sub
```

Placez ensuite le code suivant dans le module ContratsAuteur :

```
1: Sub MettreTableauAJour()
2:   Dim derligne As Integer
3:   Set MonTableauWord = GetObject("C:\Mes documents\Contrats Auteurs.doc")
4:   MonTableauWord.Tables(1).Rows.Add
5:   derligne = MonTableauWord.Tables(1).Rows.Count
6:   With MonTableauWord.Tables(1)
7:     .Cell(derligne, 1).Range.InsertAfter MesConditionsAuteur.ISBN
8:     .Cell(derligne, 2).Range.InsertAfter MesConditionsAuteur.Titre
9:     .Cell(derligne, 3).Range.InsertAfter MonAuteur.Nom
10:    If MesConditionsAuteur.TauxVariable = True Then
11:      .Cell(derligne, 4).Range.InsertAfter MesConditionsAuteur.
        ➡TauxDroitsNum1 & _
12:      " sur " & MesConditionsAuteur.NumExemplairesTaux1 & _
13:      " puis " & MesConditionsAuteur.TauxDroitsNum2
14:    Else
15:      .Cell(derligne, 4).Range.InsertAfter MesConditionsAuteur.
        ➡TauxDroitsNum1
16:    End If
17:    .Cell(derligne, 5).Range.InsertAfter MesConditionsAuteur.Avance
        ➡SurDroits
18:    .Cell(derligne, 6).Range.InsertAfter Date
19:    .Cell(derligne, 7).Range.InsertAfter MesConditionsAuteur.remise
```

```
20:         .Cell(derligne, 8).Range.InsertAfter MesConditionsAuteur.Parution
21:     End With
22:     MonTableauWord.Save
23:     MonTableauWord.Close
24: End Sub
```

Ligne 3, la variable `MonTableauWord` se voit affecter un document Word. Ligne 4, une ligne est ajoutée au premier tableau de ce document, et le numéro de cette ligne est stocké dans la variable `DerLigne` à la ligne suivante. Lignes 6 à 21, une structure `With...End With` sert à compléter les cellules du tableau avec les valeurs appropriées. Lignes 22 et 23, le tableau est enregistré puis fermé.

Conseil

Telle qu'elle se présente, cette application fonctionne. Pour la sécuriser, vous devez cependant la tester dans des conditions différentes et mettre en place des gestionnaires d'erreur. De multiples conditions peuvent générer une erreur. Ce sera par exemple le cas si un document utilisé (le modèle `Paiement.xlt` ou le document `Contrats Auteur.doc` par exemple) n'est pas disponible, ou est en cours d'utilisation par un autre utilisateur.

Annexe

Mots clés pour la manipulation de fichiers et de dossiers

Cette annexe présente les mots clés dédiés à la gestion de fichiers et de dossiers. Manipulez-les avec prudence afin de ne pas supprimer malencontreusement des fichiers importants !

Mots clés pour la manipulation des fichiers et des dossiers

<i>Mot clé</i>	<i>Description</i>	<i>Exemple</i>
<i>path</i>	Change de dossier. Notez que le lecteur courant n'est pas modifié.	ChDir «C:\Mes documents». Fait de C:\Mes documents le dossier actif.
<i>drive</i>	Change de lecteur.	ChDrive «D:» Fait du lecteur D: le lecteur actif.
<i>CurDir</i>	Retourne le chemin du dossier en cours.	

<i>Mot clé</i>	<i>Description</i>	<i>Exemple</i>
<i>(pathname, attributes)</i>	Retourne le nom des fichiers d'un dossier ou d'un volume. <i>pathname</i> est facultatif et peut représenter le nom du dossier et le lecteur. <i>attributes</i> est facultatif et représente les attributs des fichiers que l'on veut retourner. Additionnez les valeurs suivantes pour définir la valeur de <i>attributes</i> : – <i>vbNormal</i> ou 0 : Fichiers sans attributs (valeur par défaut). – <i>vbReadOnly</i> ou 1 : Fichiers accessibles en lecture seule et fichiers sans attributs. – <i>vbHidden</i> ou 2 : Fichiers cachés et fichiers sans attributs. – <i>vbSystem</i> ou 4 : Fichiers système et fichiers sans attributs (non disponible sur Macintosh). – <i>vbVolume</i> ou 8 : Nom de volume ; si un autre attribut valeur est précisé, la constante <i>vbVolume</i> est ignorée (non disponible sur Macintosh). – <i>vbDirectory</i> ou 16 : Dossiers et fichiers sans attributs.	<code>MyFile = Dir("*.doc", vbHidden).</code> Retourne le premier fichier possédant l'extension .doc et l'attribut «Fichier caché».
<i>source, destination</i>	Copie un fichier à un autre emplacement. <i>source</i> est le fichier à copier et <i>destination</i> l'endroit où il doit être copié. Notez que le nom du fichier copié peut être différent de celui du fichier source. Attention : une erreur est générée si vous tentez de copier un fichier ouvert.	<code>FileCopy "C:\toto.doc", "A:infos.doc"</code> Copie le fichier toto.doc sur le lecteur a: et sous le nom infos.doc.
<i>FileDateTime (pathname)</i>	Retourne les informations de date et d'heure d'un fichier.	<code>FileDateTime("C:\toto.doc")</code> Retourne la date et l'heure du fichier toto.doc.
<i>(pathname)</i>	Retourne la taille d'un fichier en octets.	<code>FileLen("C:\toto.doc")</code> Retourne la taille de toto.doc.
<i>FullName</i>	Retourne le nom complet d'un fichier (avec son chemin).	<code>ActiveDocument.FullName</code> Retourne le nom complet du document actif.

<i>Mot clé</i>	<i>Description</i>	<i>Exemple</i>
<i>(pathname)</i>	Renvoie une valeur représentant les attributs d'un fichier, d'un dossier ou d'un volume. La valeur retournée est la résultante de l'addition de ces valeurs : – <code>vbNormal</code> ou 0 : Fichier normal. – <code>vbReadOnly</code> ou 1 : Fichier en lecture seule. – <code>vbHidden</code> ou 2 : Fichier caché. – <code>vbSystem</code> ou 4 : Fichier système (non disponible sur Macintosh). – <code>vbDirectory</code> ou 16 : Dossier. – <code>vbArchive</code> ou 32 : Fichier modifié depuis la dernière sauvegarde (non disponible sur Macintosh).	
<i>path</i>	Crée un nouveau dossier.	<code>MkDir «C:\MonDossier»</code> Crée le dossier C:\MonDossier.
<i>Name</i>	Retourne le nom d'un fichier (sans son chemin).	<code>ActiveDocument.Name</code> Retourne le nom du document actif.
<i>Path</i>	Supprime un dossier.	<code>Rmdir «C:\MonDossier»</code> Supprime le dossier C:\MonDossier.
<i>pathname, attributes</i>	Modifie les attributs d'un fichier. Utilisez les mêmes valeurs que pour la fonction <code>GetAttr()</code> pour définir les attributs du fichier. Si le fichier est ouvert, une erreur est générée.	<code>SetAttr «C:\toto.doc», vbHidden + vbReadOnly</code> Affecte les attributs Fichier caché et Lecture seule au fichier toto.doc.

Index

Symboles

-, opérateur arithmétique 181
*, opérateur arithmétique 181
/, opérateur arithmétique 181
+, opérateur arithmétique 181
&, opérateur de concaténation 174
<, opérateur relationnel 211
<=, opérateur relationnel 211
=, opérateur relationnel 211
>, opérateur relationnel 211
>=, opérateur relationnel 211

A

Abs, fonction 264
Accelerator, propriété 387
Activate, événement 448
Activation (feuille de calcul) 448
ActiveCell, propriété 69, 443
ActiveSheet, propriété 443
ActiveWorkbook, propriété 443
Afficher
 barre d'outils 121
 boîte à outils 102

Explorateur
 de projet 89
 d'objets 93
fenêtre
 Code 104
 Propriétés 117
 UserForm 102
feuille 357
AfterUpdate, événement 408
Aide
 code 169
 VBA 96
 touche F1 360
Ajouter
 module 149
 procédure 153
Aligner, commande 347
Alignment, propriété 362
Ancrage 124
And, opérateur logique 252
Annuler 445
Apparence des contrôles 362
 Voir Propriétés des contrôles

Appels
 pile 299, 304
 procédures 131, 162
 Property Get 145
 Property Let 147
Application
 création 441
 hôte 5, 16, 86
 objet 18, 440
 déclaration et instanciation 440
 méthodes 445
 propriétés 443
Arguments
 facultatifs 138
 fonctions 29
 nommés 165
 passage 137
 procédure 164
 passés 131
Arrêt
 enregistrement 67
 mode 298
Asc, fonction 266
As, mot clé 138
Atn, fonction 264
Authentification
 Voir Sécurité
Automation 17, 203
AutoSize, propriété 372
AutoTab, propriété 373
AutoWordSelect, propriété 374

B

BackColor, propriété 363
BackStyle, propriété 363
Barre de défilement 340, 443
 Voir ScrollBar
Barre de formule 443
Barre d'état 443
Barre d'outils
 afficher/masquer 121
 personnaliser 321
 Visual Basic Editor 121

BeforeDragOver, événement 408
BeforeDropOrPaste, événement 408
BeforeRightClick, événement 448
BeforeUpdate, événement 408
Bibliothèque d'objets 17, 93
 Excel 95
 MSForms 94
 Office 95
 référencer 204
 VBA 95
Boîte à outils 88, 333
 afficher 102
 ajouter
 contrôles 353
 page 355
 contrôle
 CheckBox 336
 ComboBox 335
 CommandButton 338
 Frame 335
 Label 334
 ListBox 336
 MultiPage 339
 OptionButton 337
 ScrollBar 340
 SpinButton 340
 TabStrip 338
 TextBox 334
 ToggleButton 338
 personnaliser 352
Boîtes de dialogue
 afficher 235
 InputBox 235, 239
 MsgBox 241
 développer 329
 Voir Feuille
 Enregistrer
 macro 48
 sous 247
 Excel 246
 Macro 48
 Ouvrir 247
Boolean, type 24

- BorderColor, propriété 363
- BorderStyle, propriété 364
- Boucles 210
 - Do Loop 215
 - For Each...Next 224
 - For...Next 218
 - imbriquées 222
 - While...Wend 210
- Boutons
 - Voir* OptionButton
 - bascule 338
 - Voir* ToggleButton
 - commande 338
 - Voir* CommandButton
 - macro 48
 - Ne pas commenter 158
 - option 337
 - réorganiser 351
 - toupie 340
 - Voir* SpinButton
- ByRef, mot clé 138
- Byte, type de données numériques 180
- ByVal, mot clé 138, 140
- C**
- Cadre
 - Voir* Frame
- Calculate
 - événement 448
 - méthode 445
- Calcul forcé 445
 - feuille de calcul 448
- Call, mot clé 162
- Cancel, propriété 374
- Caption, propriété 364, 443
- Caractère
 - continuité de ligne 156
 - séparation 444
- Case à cocher 336
- Casse 52
- CBool, fonction 200, 267
- CByte, fonction 200, 267
- CCur, fonction 200, 267
- CDate, fonction 200, 267
- CDbl, fonction 200, 267
- CDec, fonction 200, 267
- Cells, propriété 69
- Cellule
 - active 66, 443
 - modification 448
- Centrer 350
- Chaîne de caractères
 - comparer 283
 - concaténer 276
 - variables tableau 278
 - espaces 280
 - extraire 281
 - longueur 281
 - manipuler 275
 - modifier 276
 - casse 283
 - rechercher 286
 - remplacer 282
 - type de données 22
 - variables 178
- Change, événement 409, 448
- ChDir, instruction 535
- ChDrive, instruction 535
- CheckBox, contrôle 336, 377, 429
 - valeur 367
- Chr, fonction 236, 266, 278
- CInt, fonction 200, 267
- Classes 15, 95
 - membres 93, 96
- Classeur 20, 444, 445
 - actif 443
 - calcul forcé 445
 - macros 55
 - stockage 54
- Clavier 64
- Click, événement 410
- CLng, fonction 200, 267
- Codage 36
- Code 36
 - aide 169
 - commentaires 156

- couleurs 159
- écriture 155
- exécuter 168
- fenêtre 102
 - deux volets 112
 - plusieurs 111
- mise en forme 158
- objets 105
- options
 - d'affichage 113
 - d'édition 114
- stockage 130
- Collection d'objets
 - For Each...Next 224
- Collection d'objets 15, 20
 - indice 20
 - méthode 28
- Column, propriété 69
- COM 203
- Combinaison (touches) 445
- ComboBox, contrôle 335, 366, 377, 378, 421
 - valeur 368
- CommandButton, contrôle 338, 374, 375
- Commande (enregistrer) 36
- Commenter 156
 - bouton 158
- Comparer (chaînes) 283
- Compilation 295
 - erreurs de 294
- Complément automatique des instructions,
 - option 170
- Comportement 372
 - Voir* Propriétés des contrôles
- Concaténation
 - chaînes de caractères 276
 - variables 181
- Conditions
 - imbriquées 231
 - opérateur relationnel 211
- Constantes 197
 - caractères 279
 - définition 133
 - Excel 25
 - intérêt 202
 - préfixes 24
 - stockage 173
 - Voir* Variables
 - type de données 24
- Const, mot clé 197
- Contrôles 329
 - Voir* Feuilles
 - ajouter 353
 - aligner 347
 - centrer 350
 - CheckBox 336, 377, 429
 - valeur 367
 - ComboBox 335, 366, 377, 378, 421
 - valeur 368
 - CommandButton 338, 374, 375
 - copier/coller 344
 - événements 408
 - Voir* Événements
 - focus 373
 - Frame 335
 - grouper 351
 - info-bulle 365
 - Label 334, 415
 - ListBox 336, 426
 - valeur 368
 - mise en forme 346
 - MultiPage 339
 - valeur 368
 - OptionButton 337, 388, 429
 - valeur 367
 - placer 341
 - procédure 402
 - programme 209
 - Voir* Structure de contrôle
 - propriétés 118
 - Voir* Propriétés des contrôles
 - raccourci 387
 - ScrollBar 340, 385, 386, 430
 - valeur 368
 - sélectionner 345
 - séparer 351
 - SpinButton 340, 385, 386, 432
 - valeur 368
 - supprimer 345

TabStrip 338
tabulation 373
TextBox 334, 377, 378, 379, 380, 418
 valeur 367
ToggleButton 338
 valeur 367
uniformiser
 espace inter-... 349
 taille 348
 valeur 367
ControlTipText, propriété 365
Conventions 9, 10
Copier/coller 344
Cos, fonction 264
Couleurs du code 159
CreateObject, fonction 193, 203
Créer
 macro 51
 procédure 152
CSng, fonction 200, 267
CStr, fonction 201, 267
CurDir, instruction 535
Currency, type de données numériques 181
CurrentRegion, propriété 80
Curseurs 443
Cursor, propriété 443
CVar, fonction 201, 267

D

Date
 fonction 228, 267, 268
 variable 184
DateSerial, fonction 269
DateValue, fonction 269
Day, fonction 268
DDB, fonction 270
Deactivate, événement 448
Débogage 293
 espions 302
 exemple 305
 fenêtre Exécution 301
 interroger 298
 modifier 301

 pas à pas 297
 pile des appels 304
 points d'arrêt 300
 tester 295
Déclaration
 explicite et implicite 174
 module 107
 procédures
 Function 141
 Property Get 144
 Property Let 146
 variables 174
Default, propriété 375
Défilement 382
 Voir Propriétés des contrôles
Delay, propriété 384
Déplacements
 codage VB 69
 Excel 63
Désactivation 448
Détecteur d'erreur 312
Développement environnement 86
 Voir Visual Basic Editor
Dialog
 collection 246
 objet 247
Dim, mot clé 177, 202
Dir, instruction 536
DisplayAlerts, propriété 443
DisplayFormulaBar, propriété 443
DisplayScrollBars, propriété 443
DisplayStatusBar, propriété 443
Document hôte 53
Do...Loop, structure de contrôle 215
Données
 conversion 198, 200
 stockage 173
 Voir Variables
 type 22
 Voir Types de données
 validation 198
Dossiers 91
 Feuilles 91

- manipuler 535
- Microsoft Excel Objets 91
- Modules 91
 - de classe 91
- Références 91
- Double
 - clic 448
 - type de données numériques 181
- DropButtonClick, événement 410
- Durée de vie (variables) 201

E

- Écriture du code 155
- Emplacement 393
 - Voir* Propriétés des contrôles
- EnableCancelKey, propriété 444
- Enabled, propriété 375
- End
 - mot clé 297, 298
 - propriété 81
 - Sub, instruction 43
- Enregistrer
 - commandes 36
 - macro 37, 38
 - enregistreur 36
 - sous 247
- Enter, événement 410
- EnterKeyBehavior, propriété 377
- Erreurs
 - compilation 294
 - détecteur 312
 - exécution 294
 - gestion 293, 312
 - gestionnaire 312
 - logiques 295
 - objet Err 313
 - On Error 312
- Espaces 280
- Espions 302
 - express 304
 - supprimer 304
- Étiquettes 312
- Événements 28, 106, 408
 - Activate 448
 - AfterUpdate 408
 - BeforeDragOver 408
 - BeforeDropOrPaste 408
 - BeforeRightClick 448
 - BeforeUpdate 408
 - Calculate 448
 - Change 409, 448
 - Click 410
 - Deactivate 448
 - DropButtonClick 410
 - Enter 410
 - Exit 410
 - feuille de calcul 448
 - gestion des... utilisateurs 440
 - Initialize 411
 - KeyDown 411
 - KeyPress 412
 - KeyUp 412
 - MouseDown 414
 - MouseMove 414
 - MouseUp 414
 - procédures 107, 132
 - SelectionChange 448
 - SpinDown 414
 - SpinUp 415
- Excel
 - barre
 - de défilement 443
 - de formule 443
 - d'état 443
 - boîtes de dialogue 246
 - cellule active 66
 - intitulé 443
 - message 443
 - modèle d'objets 29
 - référence
 - absolue 67, 69
 - relative 67, 78
 - sélection 64
- Exécution
 - code 168
 - erreurs 294

Voir Gestion des erreurs et débogages

fenêtre 301

impossible 298

macro 40

pas à pas 297

Exit

événement 410

mot clé 311, 312

Exp, fonction 265

Explorateur de projet 88, 89

afficher 89

ordre alphabétique 92

masquer 89

utiliser 91

Explorateur d'objets 88, 92

afficher 93

aide 96

fonctions 260

masquer 93

rechercher 98

utiliser 94

Exporter (module) 151, 458

Expressions 133

arithmétiques 23

opérateurs relationnels 211

F

False, valeur booléenne 24

Fenêtre

ancrer 124

Code 88, 102

affichage 111, 112, 113

options d'affichage 113

options d'édition 114

rechercher 109

remplacer 110

sélection 108

utiliser 105

Exécution 301

Propriétés 116, 117

afficher 117

utiliser 118

UserForm 88, 100

afficher 102

Variables locales 298

Feuille 329, 436

Voir Contrôles

afficher 357

contrôles

placement 341

propriétés 118

création et mise en forme 329, 358

créer 331

de calcul

activation 443, 448

calcul forcé 448

désactivation 448

double-clic 448

gérer les événements 448

modification 448

développement 331

dossier 91

exploiter 359

Voir Contrôles; *Voir* Propriétés

masquer 357

mettre en forme 346

position 394

raccourci 387

Fichiers 535

FileCopy, instruction 536

FileDateTime, instruction 536

FileLen, instruction 536

Filter, fonction 289

Fix, fonction 265

Focus 373

Fonctions 29

Abs 264

arguments 29

Asc 266

Atn 264

CBool 200, 267

CByte 200, 267

Ccur 200, 267

CDate 200, 265, 267

CDbl 200, 267

CDec 200, 267

Chr 236, 266, 278
CInt 200, 267
CLng 200, 267
conversion 200
Cos 264
CreateObject 193, 203
CSng 200, 267
CStr 201, 267
CVar 201, 267
 Date 267
Date 228
DateSerial 269
DateValue 269
Day 268
DDB 270
Excel 258
Exp 265
Filter 289
Fix 265
Format 266
FV 270, 271
GetObject 191, 203
Hex 266
Hour 269
InputBox 235
InStr 225
IPmt 272
IRR 271
IsArray 198
IsDate 198
IsEmpty 198
IsError 198
IsMissing 198
IsNull 198
IsNumeric 198
IsObject 198
Join 278
LBound 188
LCase 283
Left 225, 281
Len 281
Log 265
LTrim 280
Mid 281
Minute 269
MIRR 272
Month 268
MsgBox 241
Now 268
NPer 272
NPV 273
Oct 266
 personnalisées 258
Pmt 272
PPmt 272
PV 273
Rate 271
 recommandations 263
Replace 282
Resize 80
RGB 266
Right 281
Rnd 265
Round 265
RTrim 280
Second 269
Sgn 265
Sin 265
SLN 270
Sqt 265
Str 266
StrComp 284
StrConv 283
String 280
SYD 270
Tan 265
Time 268
Timer 269
TimeSerial 269
TimeValue 269
Trim 280
TypeName 199
types de données 198
UBound 188
UCase 283
Val 266

VarType 199
VBA 264
 conversion 266, 267
 date et heure 267, 268
 financières 270, 272
 mathématiques 264
Weekday 268
WeekdayName 268, 269
Year 268
Font, propriété 399
For Each...Next, structure de contrôle 224
ForeColor, propriété 365
Format, fonction 266
Formulaire 329
 Voir Feuilles
For...Next, structure de contrôle 218
Frame, contrôle 335
FullName, instruction 536
Function, procédures 141
 appels 163
 syntaxe 141
FV, fonction 270, 271

G

Gestion avancée des objets Excel 436, 449
Gestion des erreurs 293
 exemple 313
 questionnaire 312
GetAttr, instruction 537
GetObject, fonction 191, 203
GetOpenFilename, méthode 247
GetSaveAsFilename, méthode 247
GoTo, instruction 234
Grille 346
GroupName, propriété 388

H

Height, propriété 393
HelpContextID, propriété 388
Hex, fonction 266
HideSelection, propriété 377
Hiérarchie de classes 16
Hour, fonction 269

I

If...Then...Else, structure de contrôle 228
Info
 bulles 298, 365
 express automatique, option 171
Initialize, événement 411
InputBox
 fonction 235
 méthode 239
Instanciation 16
 d'un objet 15
InStr, fonction 225
Instructions
 casse 52
 ChDir 535
 ChDrive 535
 commenter 158
 complément 170
 conditionnelles 228
 If...Then...Else 228
 Select Case 233
 CurDir 535
 d'affectation 134
 de déclaration 134
 Dir 536
 End Sub 43
 exécutables 135
 FileCopy 536
 FileDateTime 536
 FileLen 536
 FullName 536
 GetAttr 537
 GoTo 234
 imbriquées 158
 MkDir 537
 modifier 301
 Name 537
 présentation 133
 retraits de ligne 158
 Rmdir 537
 SetAttr 537
 Sub 43
 sur plusieurs lignes 156

Integer, type de données numériques 180

Interface

utilisateur 329

Voir Feuille

Visual Basic Editor 88

Interruption (macro) 444

Int, fonction 265

Intitulé 334

Voir Label

barre de titre 443

IPmt, fonction 272

IRR, fonction 271

IsArray, fonction 198

IsDate, fonction 198

IsEmpty, fonction 198

IsError, fonction 198

IsMissing, fonction 198

IsNull, fonction 198

IsNumeric, fonction 198

IsObject, fonction 198

Is, opérateur relationnel 211

J

Join, fonction 278

K

KeepScrollsVisible, propriété 384

KeyDown, événement 411

KeyPress, événement 412

KeyUp, événement 412

L

Label, contrôle 334, 415

LargeChange, propriété 386

LBound, fonction 188

LCase, fonction 283

Left

fonction 225, 281

propriété 393

Libellé 334

Voir Label

Libérer (variable objet) 195

Like, opérateur relationnel 211

ListBox, contrôle 336, 426

valeur 368

Locked, propriété 378

Log, fonction 265

Long, type de données numériques 180

LTrim

fonctions 280

Len 281

M

Macros

accéder 53, 60

améliorer 47

autorisations 459

bouton 321

complémentaires 56

activer/désactiver 58

enregistrer 57

intégrées 59

date de création 38

écran 444

écrire 51

enregistrement 37, 38

Excel 67

enregistreur 36

exécution 40

heure précise 445

globales 55

instructions 43

interrompre 176

interdire 444

sauvegarde 457

sécurité 452

Voir Sécurité

signer 472

stockage 53

enregistrement 54

structure 42

utilisation 35

virus 452

Majuscules (convertir) 283

Manipuler 535

- Masquer 357
- Matrice, variables (de) 185
- MaxLenght, propriété 378
- Max, propriété 385
- Membres d'une classe 93, 96
- Menu (personnaliser) 323
- Messages d'alerte 443
- Méthodes 20, 27
 - Application 445
 - Calculate 445
 - collection 28
 - InputBox 239
 - OnKey 445
 - OnTime 445
 - OnUndo 445
- Microsoft Excel Objets, dossier 91
- Mid, fonctions 281
- Min, propriété 385
- Minuscules (convertir) 283
- Minute, fonction 269
- MIRR, fonction 272
- Mise à jour de l'écran 444
- Mise en forme
 - code 158
 - contrôles 346
- MkDir, instruction 537
- Mode Arrêt 298
- Modèle d'objets 16, 19, 29
- Modifier
 - bouton 48
 - cellule 27
 - chaînes 276
 - casse 283
 - Change 409, 448
 - commandes 4
 - contrôle 354
 - instructions 301
 - macro 42
 - objet 22, 26
 - propriétés 119
 - Resize 69, 80
 - SelectionChange 448
 - variable 200
- Module
 - affichage du code 114
 - ajouter 149
 - déclarations générales 107
 - de classe 91
 - dossier 91
 - exporter 151, 458
 - objets 105
 - présentation 130
 - procédures 106
 - supprimer 151
- Month, fonction 268
- Mot clé 43, 133
 - As 138
 - ByRef 138
 - ByVal 138, 140
 - Call 162
 - Const 197
 - définition 133
 - Dim 177, 202
 - End 297, 298
 - Exit 311, 312
 - Nothing 195
 - On Error 312
 - Optional 138
 - ParamArray 138
 - Private 137, 155, 202
 - Public 137, 155, 202
 - recherche 98
 - REM 156
 - Resume 312
 - Set 190
 - Static 137, 139, 203
 - Stop 298, 300, 301
 - Type 195
 - WithEvents 440
- Mot de passe 362
- MouseDown, événement 414
- MouseDown, propriété 388
- MouseMove, événement 414
- MousePointer, propriété 389
- MouseUp, événement 414
- MSForms, bibliothèque d'objets 94

MsgBox, fonction 241
MultiLine, propriété 379
MultiPage, contrôle 339
valeur 368

N

Name
instruction 537
propriété 361
Nom d'utilisateur 444
Nothing, mot clé 195
Now, fonction 268
NPer, fonction 272
NPV, fonction 273

O

Objets 105
Voir Explorateur d'objets
accéder 19
affecter une macro 325
aide 96
Application 18, 440
déclaration et instanciation 440
méthodes 445
propriétés 443
bibliothèque 17
caractéristiques 14
classes 15
classeur 444
code 105
collections 15, 20
For Each...Next 224
définition (propriétés) 44
Dialog 247
événements 28
explorer 92
gestion
avancée 449
événements 440
instanciation 15, 16
méthodes 27
modèles 16
présentation (des) 14, 29

propriétés 21
recherche 98
référencer 204
référentiel 20, 26
ThisWorkbook 445
variables 190
Workbook 20
Worksheet 448
Oct, fonction 266
Office
applications hôtes 16
bibliothèque 95
Offset, propriété 69
OLE Automation 203
On Error, mot clé 312
Onglets 338
Voir TabStrip et MultiPage
OnKey, méthode 445
OnTime, méthode 445
OnUndo, méthode 445
Opérateurs
arithmétiques 181
comparaison 211
concaténation 179
logiques 252
And 252
Or 252
Xor 252
relationnels 211
Option
ancrage des fenêtres 126
Base 1, instruction 186
Explicit, option 175
Optional, mot clé 138
OptionButton, contrôle 337, 388, 429
valeur 367
Ordre de tabulation 373, 391
définir 391
Or, opérateur logique 252
Ouvrir
boîte de dialogue 247
Visual Basic Editor 86

P

- Page 355
- ParamArray, mot clé 138
- Paramètres
 - fonctions 29
 - Visual Basic Editor 124
- Pas à pas, commande 298
- Passage d'arguments 131, 137, 164
 - apparition 164
 - nommés 165
 - référence 138
 - valeur 138, 140
- PasswordChar, propriété 362
- PathSeparator, propriété 444
- PERSONAL.XLSB 55
- Personnaliser
 - barres d'outils 321
 - boîte à outils 352
 - menus 323
 - raccourcis clavier 318
- PictureAlignment, propriété 396
- PicturePosition, propriété 398
- Picture, propriété 395
- PictureSizeMode, propriété 397
- PictureTiling, propriété 398
- Pile des appels 299, 304
- Plage de cellules
 - courantes 80
 - fin 81
 - nommées 83
 - redimensionner 80
 - utilisées 81
- Pmt, fonction 272
- Points d'arrêt 300
- Police des contrôles 399
 - Voir* Propriétés des contrôles
- Portée
 - des procédures 137
 - notion 154
 - variables 201
 - privée 201
 - procédure 201
 - publique 202
- PPmt, fonction 272
- Private, mot clé 137, 155, 202
- Procédures
 - appels 131, 162
 - créer 152
 - définition 105
 - déplacer 161
 - événement 28, 107
- Voir* Événements
 - application 441
 - classeur 445
 - feuille 448
- évènementielles 132, 402
- Function 141
 - appels 163
 - déclaration 141
- noms 136
- pas à pas 297
- passage d'arguments 164
- pile des appels 299
- portées 136, 137
- présentation 131
- privées 136, 137
- Property 142, 143
- Property Get, déclaration 144
- Property Let, déclaration 146
- publiques 136, 137
- quitter 166
- récurives 246
- Sub 136
 - appels (de) 162
- tester 295
- types de... 136
- Programmation orientée objet 13, 29
- Programme VBA
 - déroulement 209
- Voir* Structure de contrôle
 - instructions 133
 - lisibilité 162
 - modules 130
 - pas à pas 297
 - procédures 131
 - quitter 167
 - structure 130

Projets

- compiler 295
- dossiers constitutifs 91
- Explorateur de projet 92
- explorer 89
- réinitialiser 296
- structure 149
- tester 295
- verrouiller 459

Property Get, procédures

- appel 145
- syntaxe 144

Property Let, procédures

- appel 147
- syntaxe 146

Property, procédures 143

- appels 163

Propriétés 21

- accéder 26
- ActiveCell 69, 443
- ActiveSheet 443
- ActiveWorkbook 443
- Application 443
- Caption 443
- Cells 69
- Column 69
- contrôles 360
 - Accelerator 387
 - Alignment 362
 - AutoSize 372
 - AutoTab 373
 - AutoWordSelect 374
 - BackColor 363
 - BackStyle 363
 - BorderColor 363
 - BorderStyle 364
 - Cancel 374
 - Caption 364
 - ControlTipText 365
 - Default 375
 - Delay 384
 - Enabled 375
 - EnterKeyBehavior 377

- Font 399

- ForeColor 365

- GroupName 388

- Height 393

- HelpContextID 388

- HideSelection 377

- KeepScrollsVisible 384

- LargeChange 386

- Left 393

- Locked 378

- Max 385

- MaxLenght 378

- Min 385

- MouseIcon 388

- MousePointer 389

- MultiLine 379

- Name 361

- Picture 395

- PictureAlignment 396

- PicturePosition 398

- PictureSizeMode 397

- PictureTiling 398

- ScrollBars 382

- SelectionMargin 379

- SmallChange 386

- SpecialEffect 366

- StartUpPosition 394

- Style 366, 380

- TabIndex 391

- TabKeyBehavior 380

- TabStop 392

- Tag 393

- TextAlign 381

- Top 393

- TripleState 381

- Value 367

- Visible 369

- Width 393

- WordWrap 382

- CurrentRegion 80

- Cursor 443

- DisplayAlerts 443

- DisplayFormulaBar 443

- DisplayScrollBar 443
- DisplayStatusBar 443
- EnableCancelKey 444
- End 81
- en lecture-écriture 22
- en lecture seule 22
- fenêtre 116
- modifier 119
- Offset 69
- PasswordChar 362
- PathSeparator 444
- Range 69
- Row 69
- ScreenUpdating 444
- Select 69
- Selection 69
- ThisWorkbook 444
- UsedRange 81
- valeurs (des) 22, 45
- Protection
 - voir Sécurité
- Public, mot clé 137, 155, 202
- PV, fonction 273

R

- Raccourcis clavier
 - déplacer 65
 - macro 318
- Range, propriété 69
- Rate, fonction 271
- Rechercher
 - chaîne 286
 - fenêtre Code 109
 - texte 98
- Redimensionner (page) 80
- Référence
 - absolue 67, 69
 - boîte de dialogue 204
 - dossiers 91
 - relative 67, 78
- Référentiel d'objet 20, 26
- Réinitialiser 296
- REM, mot clé 156

- Remplacer
 - chaîne 282
 - texte 109, 110
- Réorganiser (boutons) 351
- Replace, fonction 282
- Resize, fonction 80
- Resume, mot clé 312
- Retrait
 - automatique, option 159
 - bouton 159
 - ligne (code) 158
- RGB, fonction 266
- Right, fonctions 281
- Rmdir, instruction 537
- Rnd, fonction 265
- Round, fonction 265
- Row, propriété 69
- RTrim, fonctions 280

S

- Sauvegarder (macros) 457
- ScreenUpdating, propriété 444
- ScrollBar
 - contrôle 340, 385, 386, 430
 - valeur 368
 - propriété 382
- Second, fonction 269
- Sécurité 452
 - macros
 - autoriser 459
 - sauvegarder 457
 - signer 472
 - virus 452
 - niveaux 452, 455
 - sources fiables 455
 - verrouiller 459
- Select Case, structure de contrôle 233
- Sélection
 - cellules 448
 - codage VB 69
 - lignes et colonnes 72
 - référence absolue 69, 71
- Excel 63
- outil 334
- plusieurs contrôles 345

- SelectionChange, événement 448
- SelectionMargin, propriété 379
- Selection, propriété 69
- Select, propriété 69
- SetAttr, instruction 537
- Set, mot clé 190
- Sgn, fonction 265
- Signature numérique 472
- Sin, fonction 265
- Single, type de données numériques 180
- SLN, fonction 270
- SmallChange, propriété 386
- Sortie
 - procédure 166
 - programme 167
- Souris 66
- SpecialEffect, propriété 366
- SpinButton, contrôle 340, 385, 386, 432
 - valeur 368
- SpinDown, événement 414
- SpinUp, événement 415
- Sqr, fonction 265
- Standard 122
- StartPosition, propriété 394
- Static, mot clé 137, 139, 203
- Stockage
 - données 173
 - Voir Variables*
 - macros 53
- Stop, mot clé 298, 300, 301
- StrComp, fonction 284
- StrConv, fonction 283
- Str, fonction 266
- String
 - fonction 280
 - type de données 22
 - variables 178
- Structure
 - macros 42
 - programmes VBA 130
 - projets 149
 - With...End With 48
- Structures de contrôle 209, 253
- Boîtes de dialogue 235
- boucles 210
 - Do Loop 215
 - For Each...Next 224
 - For...Next 218
 - While...Wend 210
- GoTo 234
- instructions conditionnelles 228
 - If...Then...Else 228
 - Select Case 233
- Opérateurs logiques 235
- Style, propriété 366, 380
- Sub
 - instruction 43
 - procédures 136
 - appels 162
- Supprimer
 - contrôles 345
 - espaces 280
 - module 151
- SYD, fonction 270
- Syntaxe
 - accès
 - objets 20
 - propriétés 26
 - arguments 137
 - nommés 165
- Const 197
- couleurs 159
- CreateObject 193
- Dim 177
- Do...Loop 215
- For Each...Next 224
- For...Next 218
- Function 141
- gestion des erreurs 312
- GetObject 191
- GoTo 234
- If...Then...Else 228
- InputBox 235, 239
- méthodes 27
- MsgBox 241
- Private 202

procédures
 appel 162
 événementielles 402
Property Get 144
Property Let 146
Public 202
Select Case 233
Set 190
Static 203
Type 195
vérification automatique 169
While...Wend 210
With...End With 44

T

TabIndex, propriété 391
TabKeyBehavior, propriété 380
Tableaux 138
 instruction Option Base 186
TabStop, propriété 392
TabStrip, contrôle 338
Tag, propriété 393
Tan, fonction 265
TextAlign, propriété 381
TextBox, contrôle 334, 377, 378, 379, 380, 418
 valeur 367
ThisWorkbook
 objet 445
 propriété 444
Time, fonction 268
Timer, fonction 269
TimeSerial, fonction 269
TimeValue, fonction 269
ToggleButton, contrôle 338
 valeur 367
Top, propriété 393
Touches, combinaison 445
Trait de soulignement 156
Trim, fonctions 280
TripleState, propriété 381
True, valeur booléenne 24
Type, mot clé 195

TypeName, fonction 199
Types de données 178
 chaîne de caractères 22
 constantes 24
 conversion 200
 valeur
 booléenne 24
 numérique 23
 validation 198
 vérifier 198

U

UBound, fonction 188
UCase, fonction 283
Underscore 156
Uniformiser
 espace, commande 349
 taille, commande 348
UsedRange, propriété 81
UserForm, fenêtre 100

V

Valeurs
 booléennes 24
 variables 183
 des contrôles 367
 numériques
 type de données 23
 variables 180
Val, fonction 266
Value, propriété 367
Variables 173, 203
 boolean 183
 concaténation 181
 conversion du type 200
 Dates 184
 déclarer 174
 définir 199
 de matrice
 rechercher 289
 durée de vie 201
 forcer la déclaration 175
 info-bulles 298

- locales 140
 - fenêtre 298
- matrice 185
 - concaténer 278
- numériques 180
- objets 190
 - libérer 195
- personnalisées 195
- portées 201
- statiques 139, 203
- String 178
 - longueur 179
- types 178
- validation 198
- Variant 185
- vérifier 198
- Variant, variables 185
- VarType, fonction 199
- VBA, bibliothèque d'objets 95
- Vérification (syntaxe) 169
- Verrouiller un projet 459
- Virus, macros 452
- Visible, propriété 369
- Visual Basic
 - mots clés 43, 133
 - présentation 13, 29
 - rechercher 98
 - structure 130
- Visual Basic Editor 129, 169
- Voir Outils VBA; Voir Programmes VBA*
- ancrer les fenêtres 124
- barres d'outils 121
- développer dans 129, 152
- environnement 88
- environnement de travail 126,
- Explorateur
 - de projet 89
 - d'objets 92

- fenêtre
 - Code 102
 - Propriétés 116
 - UserForm 100
- lancer 86
- paramétrer 124
- quitter 88

W

- Weekday, fonction 268
- WeekdayName, fonction 268, 269
- While...Wend, structure de contrôle 210
- Width, propriété 393
- With...End With, structure 44, 48
- WithEvents, mot clé 440
- WordWrap, propriété 382
- Workbook
 - événements 445
 - objets 20, 445
- Worksheet
 - événements 448
 - objets 448

X

- Xor, opérateur logique 252

Y

- Year, fonction 268

Z

- Zone de liste 336
 - Voir ListBox*
- Zone de liste modifiable 335
 - Voir ComboBox*
- Zone de texte 334
 - Voir TextBox*

Le Programmeur

Microsoft®

Excel® et VBA

Vous souhaitez automatiser vos manipulations sous Excel pour éviter d'avoir à refaire toujours les mêmes tâches ? De la simple macro enregistrée à la conception de programmes plus élaborés, cet ouvrage vous explique comment améliorer votre productivité ! Prenant en compte l'évolution d'Office, il vous aidera à développer des macros compatibles avec toutes les versions d'Excel (de 1997 à 2010).

Vous découvrirez en détail les multiples outils de Visual Basic Editor et apprendrez à déboguer vos programmes et à gérer les erreurs. Vous saurez personnaliser l'environnement d'Excel à l'aide de programmes capables de détecter tout événement utilisateur. Enfin, le dernier chapitre vous propose de réviser l'ensemble des connaissances acquises en réalisant une application complète, de sa conception à son débogage.

Qu'il s'agisse de faire face à un besoin immédiat ou de créer des programmes durables, cet ouvrage vous permettra de tirer pleinement profit d'Excel.

Les codes sources du livre sont disponibles sur www.pearson.fr.

TABLE DES MATIÈRES

- Notions fondamentales de la programmation orientée objet
- Premières macros
- Déplacement et sélection dans une macro Excel
- Découvrir Visual Basic Editor
- Développer dans Visual Basic Editor
- Variables et constantes
- Contrôler les programmes VBA
- Fonctions Excel et VBA
- Manipulation des chaînes de caractères
- Débogage et gestion des erreurs
- Intégrer des applications VBA dans l'interface d'Excel
- Créer des interfaces utilisateur
- Exploiter les propriétés des contrôles ActiveX
- Maîtriser le comportement des contrôles
- Programmer des événements Excel
- Protéger et authentifier des projets VBA
- Exemple complet d'application Excel
- Mots clés pour la manipulation de fichiers et de dossiers

Niveau : Débutant / Intermédiaire

Catégorie : Programmation et bureautique
Configuration : PC

À propos de l'auteur...

Éditeur et développeur indépendant, Mikaël Bidault a développé des modules VBA pour des maisons d'édition et des sites Internet réputés. Il a écrit de nombreux ouvrages sur ce sujet, comme *L'Intro Création de macros avec VBA pour Office* (CampusPress, 2002) et *Le Tout en Poche Excel et VBA* (CampusPress, 2006).

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 : 978-2-7440-4158-7

