

Mikaël Bidault

Couvre
Excel 2019
et
Office 365

Programmation **EXCEL** avec **VBA**

Compatible avec toutes les versions d'Excel

2^e édition



● Éditions
EYROLLES

Programmation EXCEL avec VBA

2^e édition

Maîtrisez la programmation VBA et tirez pleinement profit d'Excel. Cet ouvrage détaille les principes de la programmation orientée objet, le langage VBA et Visual Basic Editor, l'environnement de programmation d'Excel.

Intégrez pleinement vos programmes dans le ruban d'Excel. Vous apprendrez à créer des onglets personnalisés pour vos programmes VBA avec Custom UI Editor.

« **EXCELliez** ». Le dernier chapitre offre l'occasion de réviser l'ensemble des connaissances acquises lors de votre lecture, en développant un programme complet de génération de feuilles de paie Excel qui interagit avec Word.

Distribuez vos applications Excel de façon professionnelle. Créez un programme d'installation avec Inno Setup pour distribuer vos applications Excel simplement.

Compléments web

Tous les exemples des programmes du livre sont en téléchargement sur notre site Internet www.editions-eyrolles.com/dl/0067786.

À qui s'adresse cet ouvrage ?

- Aux utilisateurs d'Excel désireux d'améliorer leur productivité
- Aux développeurs qui souhaitent créer et distribuer des solutions Excel sûres et efficaces
- Aux personnes qui veulent s'initier à la programmation via le tableur de Microsoft

Au sommaire

Découvrir la programmation Excel. Notions fondamentales de la programmation orientée objet (POO) • Premières macros • Déplacement et sélection dans une macro Excel • Découvrir Visual Basic Editor • **Programmer en Visual Basic.** Développer dans Visual Basic Editor • Variables et constantes • Contrôler les programmes VBA • Fonctions Excel et VBA • Manipuler des chaînes de caractères • Déboguer et gérer les erreurs • Intégrer des applications VBA dans l'interface d'Excel • **Développer des interfaces utilisateur.** Créer des interfaces utilisateur • Exploiter les propriétés des contrôles • Maîtriser le comportement des contrôles • **Notions avancées de la programmation Excel.** Programmer des événements Excel • Protéger et authentifier des projets VBA • Exemple complet d'application Excel. **Créer un programme d'installation pour distribuer vos applications Excel.**

Éditeur et développeur indépendant, **Mikaël Bidault** développe des compléments Word et Excel pour différentes sociétés. Il est le créateur de l'Articho, un complément VBA pour Word dédié à l'édition print et numérique (www.articho.eu).

www.editions-eyrolles.com

Programmation Excel avec VBA

2^e édition

DANS LA MÊME COLLECTION

É. SARRION. – **React.js.**

N° 67756, 2019, 358 pages.

R. GOETTER. – **Grid Layout.**

N° 67683, 2019, 144 pages.

C. BLAESS. – **Solutions temps réel sous Linux.**

N° 67711, 3^e édition, 2019, 320 pages.

C. PIERRE DE GEYER, J. PAULI, P. MARTIN, E. DASPET. – **PHP 7 avancé.**

N° 67720, 2^e édition, 2018, 736 pages.

H. WICKHAM, G. GROLEMUND. – **R pour les data sciences.**

N° 67571, 2018, 496 pages.

F. PROVOST, T. FAWCETT. – **Data science pour l'entreprise.**

N° 67570, 2018, 370 pages.

J. CHOKOGOUE. – **Maîtrisez l'utilisation des technologies Hadoop.**

N° 67478, 2018, 432 pages.

H. BEN REBAH, B. MARIAT. – **API HTML 5 : maîtrisez le Web moderne !**

N° 67554, 2018, 294 pages.

W. MCKINNEY. – **Analyse de données en Python.**

N° 14109, 2015, 488 pages.

E. BIERNAT, M. LUTZ. – **Data science : fondamentaux et études de cas.**

N° 14243, 2015, 312 pages.

SUR LE MÊME THÈME

D.J. DAVID. – **VBA pour Excel 2010, 2013 et 2016.**

N° 14457, 2016, 324 pages.

N. BARBARY. – **Excel expert.**

N° 13692, 2^e édition, 2014, 444 pages.

J.-M. LAGODA, F. ROSARD. – **Réaliser des graphiques avec Excel.**

N° 56425, 2016, 128 pages.

Retrouvez nos bundles (livres papier + e-book) et livres numériques sur

<http://izibook.eyrolles.com>

Mikaël Bidault

Programmation Excel avec VBA

Compatible avec toutes les versions d'Excel

2^e édition

● Éditions
EYROLLES

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

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

© Groupe Eyrolles, 2017.
© Éditions Eyrolles, 2019, ISBN : 978-2-212-67786-7

Table des matières

Introduction	1
Compléments VBA et compléments Office	2
VBA, pour quoi faire ?	2
Des programmes	4
Une application hôte et des projets	5
Un langage de programmation	5
Un environnement de travail	7
Conventions typographiques	8
Codes sources des exemples du livre	9
PREMIÈRE PARTIE	
Découvrir la programmation Excel	11
CHAPITRE 1	
Notions fondamentales de la programmation orientée objet (POO)	13
Comprendre le concept d'objet	13
Objets et collections d'objets	14
Application hôte et modèles d'objets	15
Accéder aux objets	18
Les propriétés	20
Les méthodes	25
Les événements	26
Les fonctions	26
Le modèle d'objets d'Excel	27

CHAPITRE 2

Premières macros	31
Créer une macro GrasItalique	32
Afficher l'onglet Développeur	32
Démarrer l'enregistrement	33
Enregistrer les commandes de la macro	35
Exécuter la macro	35
Structure de la macro	36
Améliorer la macro	41
Une autre méthode d'enregistrement	43
Enregistrement	43
Structure de la macro	43
Écrire la macro	44
Exécution de la macro	45
Choisir l'accessibilité des macros	46
Accessibilité globale ou limitée	46
Classeurs et modèles	47
Le classeur de macros personnel	47
Les macros complémentaires	48
Définir le classeur de stockage lors de l'enregistrement d'une macro	50
Accéder aux macros d'un classeur spécifique	52

CHAPITRE 3

Déplacement et sélection dans une macro Excel	55
Méthodes de sélection dans une feuille Excel	56
Clavier	56
Souris	57
Notion de cellule active	57
Références relatives et références absolues	58
Coder les déplacements effectués lors de l'enregistrement d'une macro ...	59
Référence absolue aux cellules	60
Référence relative aux cellules	67
Référence aux cellules en fonction de leur contenu	69
Référence aux plages de cellules nommées	71

CHAPITRE 4

Découvrir Visual Basic Editor	73
Accéder à Visual Basic Editor	73
Les outils et les fenêtres de Visual Basic Editor	76
L'Explorateur de projet	77
L'Explorateur d'objets	80
La fenêtre UserForm	86
La fenêtre Code	89
La fenêtre Propriétés	101
Les barres d'outils	105
Paramétrer Visual Basic Editor	108

DEUXIÈME PARTIE

Programmer en Visual Basic	111
---	-----

CHAPITRE 5

Développer dans Visual Basic Editor	113
Structure des programmes Visual Basic	113
Les modules	113
Les procédures	114
Les instructions	116
Les différents types de procédures	118
Procédures Sub	118
Procédures Function	122
Procédures Property	124
Des projets bien structurés	129
Ajouter un module	129
Supprimer un module	130
Créer une procédure	132
Écrire l'instruction de déclaration	132
La boîte de dialogue Ajouter une procédure	133
La notion de portée	134
Écriture et mise en forme du code	135
Déplacer une procédure	140

Appel et sortie d'une procédure	140
Appel d'une procédure Sub	140
Appels de procédures Function et Property	142
Passage d'arguments	142
Sortie d'une procédure	144
Sortie d'un programme	145
Exécuter du code	146
Aide à l'écriture de code	146
Vérification automatique de la syntaxe	146
Complément automatique des instructions	147
Info express automatique	148
CHAPITRE 6	
Variables et constantes	149
Déclarer une variable	149
Déclaration implicite	149
Déclaration explicite	150
Types de données des variables	153
Chaînes de caractères	153
Valeurs numériques	155
Valeurs booléennes	157
Dates	157
Type Variant	158
Variables de matrice	159
Variables objets	163
Types de données personnalisés	167
Constantes	168
Validation et conversion des types de données	169
Portée et durée de vie des variables	172
Portée de niveau procédure	172
Portée de niveau module privée	172
Portée de niveau module publique	173
Variables statiques	173
Traitement entre applications à l'aide de variables objets	173

CHAPITRE 7

Contrôler les programmes VBA	177
Répéter une série d'instructions : les boucles	177
La boucle While...Wend	178
La boucle Do...Loop	181
La boucle For...Next	185
La boucle For Each...Next	189
Utiliser des instructions conditionnelles	192
La structure de contrôle If...Then...Else	193
La structure de contrôle Select Case	197
Définir l'instruction suivante avec GoTo	198
Interagir avec l'utilisateur via des boîtes de dialogue	198
La fonction InputBox	199
La méthode InputBox	202
La fonction MsgBox	204
Affichage de boîtes de dialogue Excel	209
Utiliser les opérateurs logiques	213
Trier des données	214

CHAPITRE 8

Fonctions Excel et VBA	217
Utiliser les fonctions Excel dans VBA	217
Créer des fonctions Excel personnalisées	218
Intégrer une fonction via l'Explorateur d'objets	219
Insérer une fonction VBA dans votre code	219
Insérer une fonction Excel dans votre code	220
Recommandations pour l'écriture de fonctions Excel	221
Les limites de la cellule	221
Principales fonctions VBA	222

CHAPITRE 9

Manipuler des chaînes de caractères	229
Modifier des chaînes de caractères	229
Concaténer des chaînes	229
Insérer des caractères non accessibles au clavier	231
Répéter une série de caractères	232
Supprimer les espaces superflus d'une chaîne	233
Extraire une partie d'une chaîne	233
Effectuer des remplacements au sein d'une chaîne	234
Modifier la casse des chaînes de caractères	235
Comparer des chaînes de caractères	236
Rechercher dans les chaînes de caractères	237
Rechercher une chaîne dans une chaîne	237
Scinder une chaîne	240
Rechercher une chaîne dans une variable de matrice	241

CHAPITRE 10

Déboguer et gérer les erreurs	245
Les étapes et les outils du débogage	245
Test du projet	246
Exécuter pas à pas	248
La fenêtre Variables locales	249
Les points d'arrêt	250
Modifier l'ordre d'exécution des instructions	251
La fenêtre Exécution	252
Les espions	252
La pile des appels	254
Exemple de débogage	255
Recherche du bogue	256
Résolution du bogue	258
Gestion des erreurs et des exceptions	261
Exemple de gestion d'erreur	262

CHAPITRE 11

Intégrer des applications VBA dans l'environnement d'Excel	265
Affectations et exécution de macros	266
Affecter un raccourci clavier à une macro	266
Affecter une macro à un bouton de commande	267
Affecter une macro à un objet.	268
Exécuter une macro à partir de la barre d'outils Accès rapide	270
Exécuter une macro à partir du ruban.	272
Créer un onglet personnalisé avec Custom UI Editor for Microsoft Office . .	274
Les exemples de fichiers XML de Custom UI Editor.	275
Analyse du fichier XML Custom Tab.	277
Validation d'un fichier XML.	279
Affectation de macros aux contrôles du ruban : les Callbacks	280
Les contrôles disponibles pour le ruban	283
La banque d'images Microsoft Office.	285
Création de l'onglet Audit	286

TROISIÈME PARTIE

Développer des interfaces utilisateur	301
--	------------

CHAPITRE 12

Créer des interfaces utilisateur.	303
Les phases de développement de feuilles	303
Créer une feuille	304
Les contrôles de la boîte à outils	306
Outil Sélection.	306
Contrôle Label.	306
Contrôle TextBox	307
Contrôle ComboBox	307
Contrôle Frame	308
Contrôle ListBox	308
Contrôle CheckBox	309
Contrôle OptionButton	310
Contrôle ToggleButton	310

Contrôle CommandButton	310
Contrôle TabStrip	311
Contrôle MultiPage	311
Contrôle ScrollBar	312
Contrôle SpinButton	313
Placer des contrôles sur une feuille	313
Copier-coller des contrôles	315
Sélectionner plusieurs contrôles	316
Supprimer des contrôles	317
Mise en forme des contrôles	317
La grille	317
Aligner les contrôles	319
Uniformiser la taille des contrôles	320
Uniformiser l'espace entre les contrôles	320
Centrer les contrôles	321
Réorganiser les boutons de commande	322
Grouper ou séparer des contrôles	322
Personnaliser la boîte à outils	323
Ajouter/supprimer un contrôle	323
Ajouter/supprimer une page	326
Afficher/masquer une feuille	328

CHAPITRE 13

Exploiter les propriétés des contrôles	331
Propriété Name	332
Apparence	333
Alignment	333
BackStyle	334
BackColor	334
BorderStyle	335
BorderColor	335
Caption	335
ControlTipText	336
ForeColor	336
SpecialEffect	336

Style	337
Value	337
Visible	339
Comportement	341
AutoSize	341
AutoTab	342
AutoWordSelect	343
Cancel	343
Default	344
Enabled	344
EnterKeyBehavior	346
HideSelection	346
Locked	347
MaxLength	347
MultiLine	347
SelectionMargin	348
Style	348
TabKeyBehavior	349
TextAlign	349
TripleState	349
WordWrap	350
Défilement	351
ScrollBars	351
KeepScrollsVisible	352
Delay	352
Max et Min	353
SmallChange	353
LargeChange	354
Divers	354
Accelerator	354
GroupName	355
HelpContextID	355
MouseIcon	356
MousePointer	356
TabIndex	358
TabStop	359
Tag	359

Emplacement	360
Height et Width	360
Left et Top	360
StartPosition	360
Image	362
Picture	362
PictureAlignment	363
PicturePosition	364
PictureSizeMode	364
PictureTiling	364
Police	365
Font	365

CHAPITRE 14

Maîtriser le comportement des contrôles	367
Créer des procédures événementielles	367
Créer une procédure	367
Les événements	373
Exemples d'exploitation des contrôles	378
Label	378
Contrôle TextBox	380
ComboBox	382
ListBox	387
CheckBox et OptionButton	390
ScrollBar	390
SpinButton	392
Exploiter les informations d'une feuille VBA	394

QUATRIÈME PARTIE

Notions avancées de la programmation Excel..... 397

CHAPITRE 15

Programmer des événements Excel..... 399**L'objet Application..... 399**

Déclaration et instanciation de l'objet Application..... 400

Création de procédures événementielles de niveau application..... 401

Propriétés de l'objet Application..... 402

Méthodes de l'objet Application..... 403

L'objet ThisWorkbook..... 404**L'objet Worksheet..... 406**

CHAPITRE 16

Protéger et authentifier des projets VBA..... 409**Les virus macros..... 409****Se protéger des virus macros..... 409**

Définir un niveau de sécurité..... 410

Les signatures numériques..... 412

Sauvegarder des macros..... 413

Protéger l'accès aux macros..... 414

Verrouiller un projet..... 414

Limiter les droits d'exécution d'une macro..... 415

Authentifier ses macros..... 425

Obtenir une authentification..... 425

Authentifier une macro..... 426

CHAPITRE 17

Exemple complet d'application Excel..... 427**Présentation du projet d'application Excel..... 427**

Avant de commencer..... 428

Identification des informations à recueillir..... 429

Définition de la structure du programme..... 430

Créer un modèle Excel	435
Définir et créer des interfaces	437
Feuille fmContratAuteur	437
Feuille fmContratConditions	448
Feuille fmContratDates	460
Feuille fmContratImpression	465
Feuille fmContratFin	470
Écriture des procédures d'édition de documents	471
Édition des feuilles de paie	472
Mise à jour du tableau Word	474
ANNEXE A	
Mots-clés pour la manipulation de fichiers et de dossiers	477
ANNEXE B	
Créer un programme d'installation pour distribuer vos applications Excel	479
Présentation d'Inno Setup	480
Création du programme d'installation	480
Test du programme d'installation	488
Correction du programme d'installation	491
Index	495

Introduction

Visual Basic pour Applications, VBA, est la solution de programmation intégrée aux applications de la suite Office. Grâce à VBA, l'utilisateur d'Excel tire pleinement profit du tableur de Microsoft en en développant les fonctionnalités pour ses besoins spécifiques. Maîtriser VBA, c'est à coup sûr améliorer sa productivité.

L'intégration dans Excel de Visual Basic pour Applications, un *environnement de développement intégré* complet et professionnel, remonte à sa version 97. Office 2013, Office 2016 et Office 2019 intègrent la version 7.1 de Visual Basic, tandis qu'Office 2010 propose la version 7.0 et que XP, 2003 et 2007 fournissent Visual Basic 6.3. Entre ces versions, les différences sont quasi-inexistantes.

Cet ouvrage traite de la programmation des versions 97 à 2019 d'Excel. Sauf exceptions signalées, les explications et les exemples proposés sont valides pour toutes les versions d'Excel. En effet, des unes aux autres, 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, sachez que vous pourrez appliquer les connaissances acquises lors de la lecture de ce livre, aussi bien avec Excel 2003 sous Windows XP qu'avec la version 2016 et un système Windows 10. Mieux, les programmes développés pour Excel 97 fonctionnent avec toutes les versions ultérieures du tableur et, dans la très grande majorité des cas, les programmes développés dans Excel 2016 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.

Définition

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.

Compléments VBA et compléments Office

Avec Office 2013, Microsoft a introduit un nouveau type de compléments, les compléments Office. Contrairement à ceux développés en VBA, les compléments Office ne sont pas installés sur l'ordinateur de l'utilisateur, mais hébergés sur un serveur distant à partir duquel ils s'exécutent. Ils sont développés à partir de technologies Web, telles que HTML 5, JavaScript, CSS 3, XML et des API REST.

En termes d'expérience utilisateur, les compléments Office s'apparentent à des applications mobiles auxquelles on s'abonne via l'Office store et s'exécutent systématiquement dans un panneau qui leur est dédié. Cependant, si vous souhaitez développer des solutions professionnelles dans le cadre d'une entreprise et non dans le but de les commercialiser via l'Office store, VBA reste presque toujours la solution la plus simple et la plus souple à mettre en œuvre et à déployer.

VBA, pour quoi faire ?

Excel offre des possibilités très étendues. Pourtant, quelle que soit la puissance de ses fonctions, 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 selon 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 – par exemple, une fonction personnalisée qui calcule 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 et à des commandes d'onglets afin d'en améliorer l'accessibilité.
- 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.

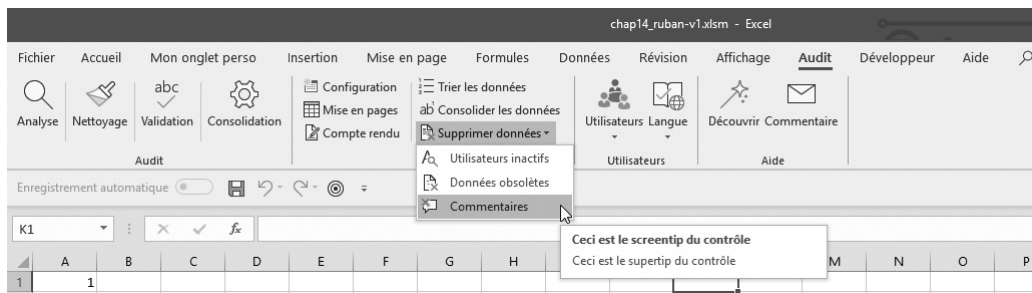


Figure 1 – Vous pouvez affecter des macros VBA à des commandes sur des onglets personnalisés.

- 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 sait 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 courriel 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.

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. Cependant, 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.

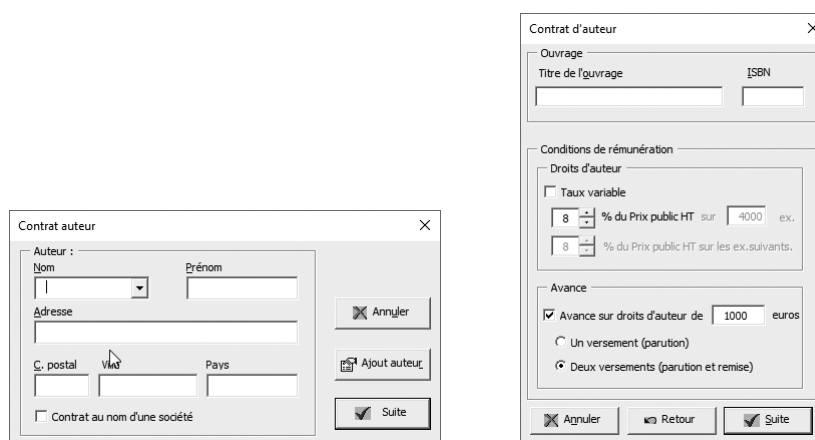


Figure 2 – Visual Basic pour Applications vous permet de développer des interfaces utilisateur évoluées.

Des programmes

Les projets VBA sont des programmes ou *macros* écrits dans le langage Visual Basic. Si vous ne possédez aucune expérience préalable en 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.

Définition

Le terme macro désigne le regroupement d'un ensemble de commandes en une seule. On parle parfois de macrocommande 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 ou 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, une solution 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. Cependant, 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 orientée objet* (POO). Nous présenterons donc les concepts de ce type de programmation. Vous apprendrez ce qu'on appelle un objet, une propriété, une méthode ou un module de classe. 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.

VBA intègre un grand nombre d'instructions, grâce auxquelles vous développerez des macros qui identifient très précisément l'état de l'application et des documents et reproduisent l'exécution de la plupart des commandes disponibles dans l'application hôte.

Vous verrez que certaines instructions sont spécifiques à Excel, par exemple celles qui affectent 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. Cependant, certaines 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, notamment celles qui règlent le comportement d'une macro : réaliser des opérations en boucle, induire des réactions face à certains paramètres, afficher des boîtes de dialogue simples (figures 3 et 4) ou développer des interfaces utilisateur évoluées (figure 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. Pour la plupart, ces instructions ne peuvent être générées par enregistrement de macros et doivent donc être éditées manuellement dans Visual Basic Editor.

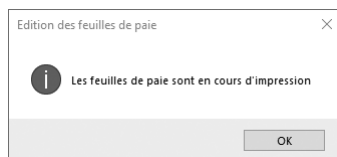


Figure 3 – La fonction VBA MsgBox affiche une boîte de dialogue.

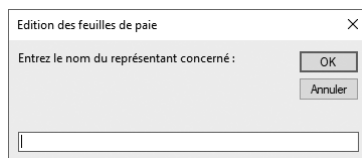


Figure 4 – Il existe une version VBA et une version Excel de la fonction InputBox.

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 exemples 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.

Définition

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 ».

Ce livre présente et illustre d'exemples commentés l'ensemble des structures de contrôle qui servent à 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 VBA fournit une référence complète du langage, facilement accessible et consultable.

Si vous n'avez aucune expérience de programmation, peut-être ce *Visual Basic* vous apparaît-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

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

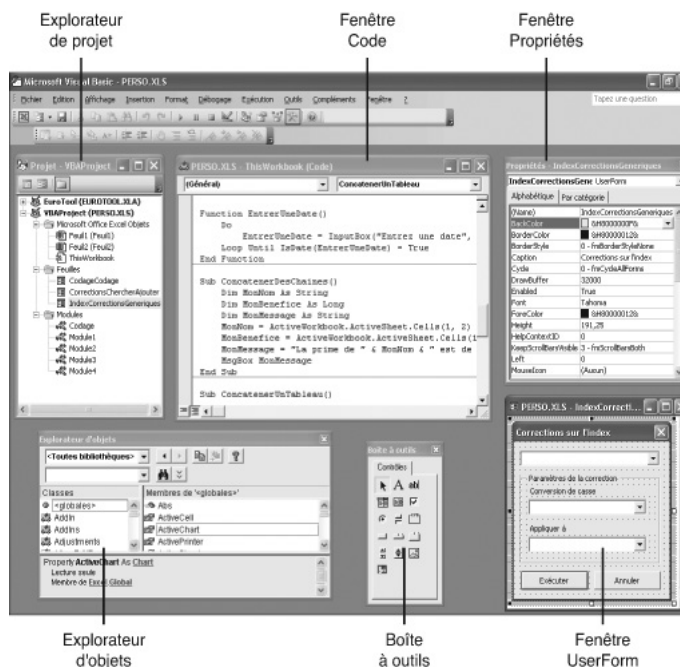


Figure 5 – Visual Basic Editor est l'environnement de développement de Visual Basic pour Applications.

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 un outil de débogage de vos projets VBA d'une grande efficacité. Il propose nombre d'outils pour tester les macros et 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, cet environnement 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.

Conventions typographiques

Afin de 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 suite de la ligne précédente.

Par ailleurs, vous rencontrerez au long de cet ouvrage différents types de notes, matérialisées par des encadrés.

Info

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

Définition

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

Attention

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

Rappel

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.

Astuce

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

Conseil

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

Codes sources des exemples du livre

Les exemples du livre sont proposés en téléchargement sur le site des éditions Eyrolles. Vous pouvez ainsi tester tous les exemples à partir des fichiers Excel qui les intègrent. Cela vous évitera également de saisir le code dans Visual Basic Editor. Avant de poursuivre la lecture de ce livre, téléchargez les exemples à l'adresse suivante : <http://www.editions-eyrolles.com/dl/0067786>.

PREMIÈRE PARTIE

Découvrir la programmation Excel

1

Notions fondamentales de la programmation orientée objet (POO)

Visual Basic est un langage de programmation *orienté objet*. En tant que tel, il repose sur des concepts communs à tous les langages de POO. 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 de 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 donner 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 POO, 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 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. Cette analogie 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 (« Regarde un peu ma belle 504 verte »). 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 un véhicule bien spécifique, en fonction de ses goûts, de ses aspirations, de ses souvenirs, etc. Cependant, en tant qu'objet *Voiture*, elle possède un certain nombre de *propriétés* (une carrosserie, des roues, un moteur) et autorise un certain nombre de *méthodes* (démarrer, freiner, tourner) 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* et 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*.

Info

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 fera découvrir les objets Excel les plus importants.

Définition

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 Peugeot 504 verte 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 verte, 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 véhicule. 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 des feuilles qui le composent, etc.)

Info

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

Le grand intérêt de la programmation orientée objet, c'est qu'il n'est pas indispensable 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), 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 leurs fonctions. 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 disponibles dans la salle de bains (savon, robinet, placard). Vous ne devriez cependant pas y trouver de brosse à dents, ni aucun des objets spécifiques à la toilette. En revanche, vous pourrez utiliser le four, ouvrir le réfrigérateur 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 dernière 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 sont capables d'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 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 accéder facilement puisque le programme ne pourra être exécuté qu'à partir de celle-ci.

Info

Tous les projets développés dans cet ouvrage seront hébergés dans Excel. Pour accéder aux objets d'une application autre que l'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. Elle est composée d'un certain nombre d'objets – constituant une *bibliothèque* – 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.

Info

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

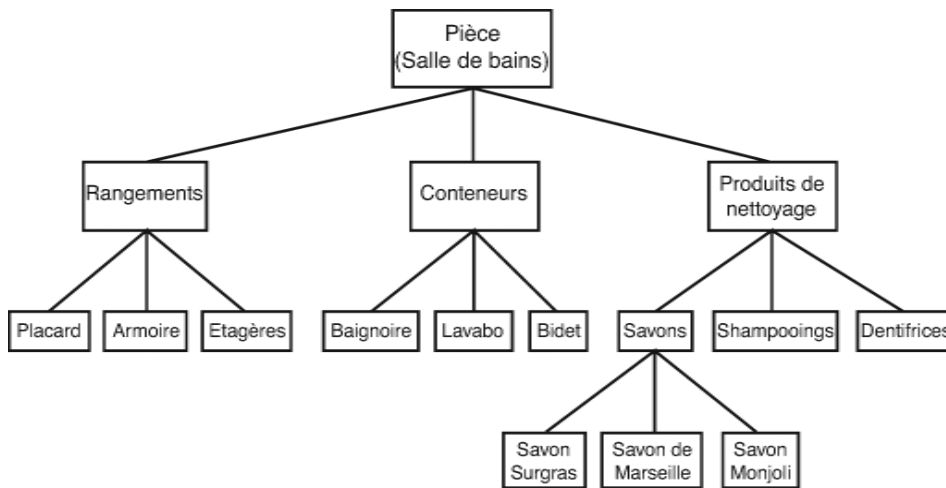


Figure 1-1 – L'ensemble des objets d'une application est structuré selon un modèle d'objets qui en définit les rapports et la hiérarchie.

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, plus les objets sont précis et donc spécifiques de la pièce ou de l'application. Dans Excel par exemple, 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 VBA à partir du menu « ? », puis sélectionnez la commande Référence VBA d'Excel. Vous apprendrez à accéder à Visual Basic Editor au prochain chapitre.

Notez que l'appartenance des objets à des branches distinctes du modèle 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*.

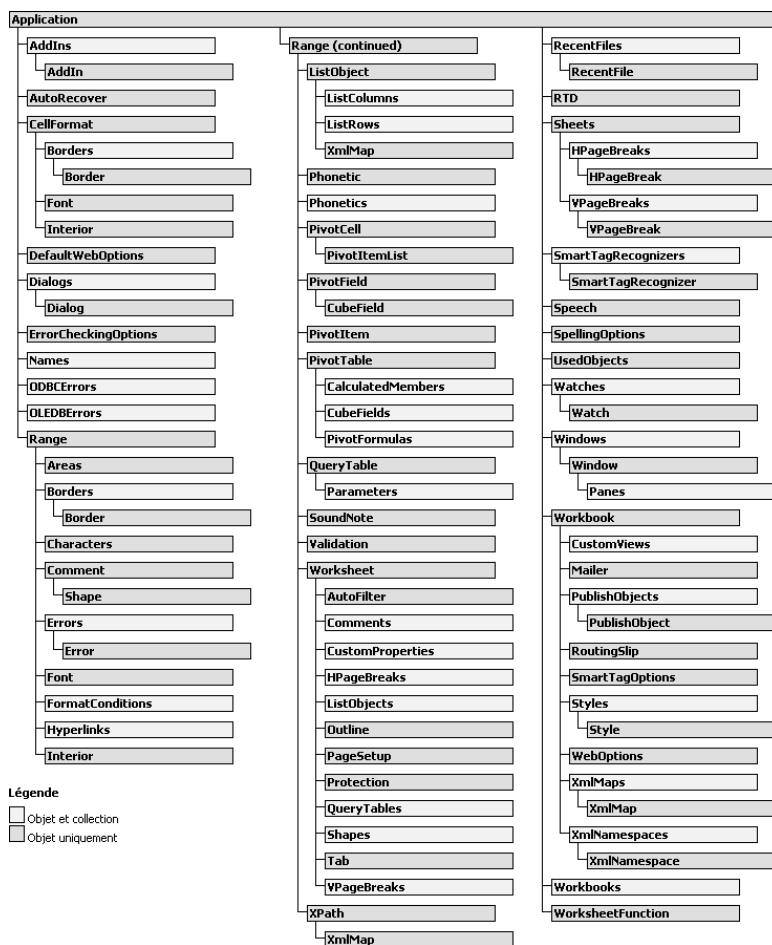


Figure 1-2 – Le modèle d'objets d'Excel.

Un objet peut en englober d'autres ; il est alors qualifié de *conteneur*. C'est le cas de l'objet `Application`, mais c'est aussi vrai pour beaucoup d'autres objets du modèle d'Excel. Par exemple, un `Workbook` contient des `Worksheet` (feuilles de calcul), contenant eux-mêmes des `Range` (cellules et plages de cellules).

Accéder aux objets

Le modèle détermine le chemin à emprunter pour accéder à un objet. Pour vous laver les dents, vous devez d'abord accéder à votre brosse à dents. Même si le processus est inconscient, vous identifiez l'objet *Brosse à dents* par son emplacement : il est situé dans la salle de bains, parmi les

objets et produits de toilette. De la même façon, en Visual Basic, vous devez identifier un objet avant de pouvoir agir dessus (appliquer l'une de ses méthodes ou modifier la valeur de l'une de ses propriétés). Lorsque vous souhaitez vous laver les dents, vous pensez et suivez inconsciemment les étapes suivantes :

- aller à la salle de bains ;
- se diriger vers les produits de toilette ;
- choisir parmi ceux-ci la brosse à dents et le dentifrice et s'en saisir.

Pour accéder à un objet Excel, vous opérerez selon le même mode, c'est-à-dire en partant du haut de la hiérarchie 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_Objet")
```

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

```
Piece.ProduitsHygiene("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 activant 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`.

Info

Outre son nom, chaque objet est identifié par une valeur d'indice représentant sa position dans la collection. Cette valeur peut être utilisée pour renvoyer un objet précis 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 la liste des fenêtres du bouton `Changer de fenêtre` de l'onglet `Affichage`.

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 regardez déjà 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 `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
```

Info

Une petite finesse sémantique : 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 des objets `Voitures` y sont définies. Les objets ou classes situés immédiatement sous `Voitures` dans le modèle 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 `Vert`, 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. Les propriétés dont les valeurs peuvent être changées sont dites en lecture-écriture.

Certaines propriétés ne sont pas modifiables et sont dites en lecture seule. Vous pouvez, par exemple, modifier la propriété `Etat_du_moteur` (allumé ou éteint) d'un objet `Voiture`, mais pas sa propriété `Marque`. Il est possible de changer le nombre de feuilles qui composent un classeur, mais pas 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, en supposant que le litre d'essence est à 1,50 euros, si la valeur BMW est affectée à la propriété `Marque` d'un objet `Voiture` et la valeur 40 (litres) affectée à sa propriété `Contenu_Réservoir`, vous ferez un plein à 60 euros. Si les valeurs Citroën et 2CV sont respectivement affectées aux propriétés `Marque` et `Modèle` et si la propriété `Contenu_Réservoir` a une valeur égale à 20, vous ne ferez qu'un plein à 30 euros.

Types de valeurs des propriétés

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

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

Chaîne 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 peut contenir jusqu'à environ deux milliards de caractères. En Visual Basic, les chaînes sont placées entre guillemets :

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

Les chaînes 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.

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.

Valeur numérique

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

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

Info

Le caractère E dans une variable numérique signifie « exposant » et représente une puissance de 10. 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. Leur combinaison détermine combien de kilomètres peuvent être parcourus avant la panne sèche, selon l'expression arithmétique suivante :

```
Kilomètres_Avant_Panne_Sèche = Contenu_Réservoir / Consommation
```

Info

Notez qu'une expression arithmétique peut être composée de nombres (100 / 25), de variables auxquelles sont affectées des valeurs numériques (nombre1 + nombre2), ou d'une combinaison des deux (nombre1 - 25). Les variables sont étudiées au chapitre 6.

Les valeurs numériques pouvant être affectées à une propriété varient selon les propriétés et les objets. Dans Excel par exemple, 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.

Valeur booléenne

Certaines propriétés ne peuvent prendre que deux états : elles sont vérifiées ou elles ne le sont pas. Elles sont attachées à une valeur de type `Boolean`, ou valeur booléenne, qui vaut `True` ou `False`.

La propriété `Moteur_Allumé` d'un objet `Voiture` est une valeur booléenne : `True` si le moteur de l'objet `Voiture` est allumé, `False` dans le cas contraire.

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 `Saved`, qui renvoie `True` si aucune modification n'a été apportée au document depuis son dernier enregistrement, ou `False` dans le cas contraire.

Info

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.

Constante

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 derniers sont souvent des constantes et 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, les constantes associées se retrouvent aussi dans toutes les applications Office. Une constante intégrée de VBA commence par deux lettres en minuscules indiquant l'application à laquelle elle appartient. Le tableau suivant 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
fm	Feuilles Visual Basic

Lorsqu'une propriété accepte des constantes pour valeurs, leur nombre 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 `État` d'un objet `Clignotant` pourrait donc accepter l'une des quatre constantes `Clignotant` suivantes :

```
ClignotantAucun  
ClignotantDroite  
ClignotantGauche  
ClignotantWarning
```

Excel intègre de nombreuses 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 et 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* correspondrait par exemple 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.

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* est la valeur que vous souhaitez lui affecter.

Le type de la valeur (chaîne, nombre, constante ou booléen) 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

Syntaxe	Type de la valeur affectée	Conséquence pour l'objet <i>Voiture</i>
<code>Voitures("MaVoiture").Immatriculation = "BS606XH"</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	Le clignotant est en position Warning.

Pour mémoriser la valeur d'une propriété d'un objet donné, on la stocke généralement 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 et elles dépendent étroitement de l'objet. Les objets de la classe `Voitures` disposent de 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 commandes communes aux applications. Par exemple, `Open` (ouvrir) et `Close` (fermer) s'appliquent aussi bien à un classeur Excel qu'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 sera peut-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 pour appliquer une méthode à un objet est la suivante :

`Expression.Méthode`

où `Expression` est une expression renvoyant un objet – un référentiel d'objet – et `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 :

`Collection.Méthode`

Vous pouvez, par exemple, arrêter tous les objets de la collection `Voitures` :

`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 dans une collection. La méthode utilisée est alors généralement *Add* – l'équivalent Visual Basic de l'onglet Fichier. Par exemple, pour créer un nouveau classeur Excel, vous ferez appel à `Workbooks.Add`.

Vous définissez ensuite 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 et qui déclenche l'exécution d'un programme lorsqu'elle 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.

Définition

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* reconnaîtront par exemple l'événement *Choc*, dont la détection entraînera l'ouverture de l'objet *Airbag*, autrement dit l'application de la méthode *Ouvrir* à ce dernier.

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 exploitables directement. Par exemple, *Asc* renvoie le code ASCII du caractère sélectionné, tandis que *Int* renvoie la partie entière d'un nombre. Certaines fonctions sont particulièrement utiles. C'est le cas de *MsgBox*, qui affiche une boîte de dialogue contenant des boutons (tels que *Oui*, *Non* ou *Annuler*) et qui renvoie une valeur reflétant le choix de l'utilisateur.

Vous créerez aussi vos propres fonctions pour traiter les valeurs qui leur seront passées et 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'instructions 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'Excel

Collection (objet)	Description
Objets de niveau <i>Application</i>	
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 <i>Application</i> . Un objet Name représente un nom défini pour une plage de cellules
Windows (Window)	L'ensemble des fenêtres disponibles (accessibles via le bouton Changer de fenêtre de l'onglet Affichage)
Workbooks (Workbook)	L'ensemble des classeurs ouverts
Worksheetfunction	On utilise l'objet <code>Worksheetfunction</code> pour accéder aux fonctions de feuilles de calcul à partir de VBA. Faites suivre <code>Worksheetfunction</code> d'un point, puis du nom de la fonction et de ses arguments entre parenthèses
Objets de l'objet <i>Workbook</i>	
Charts (Chart)	L'ensemble des feuilles graphiques de l'objet <i>Workbook</i>
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 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 <i>Workbook</i> désigné
Objets de l'objet <i>Worksheet</i>	
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 <i>Worksheet</i> désigné
HPageBreaks (HPageBreak)	Les sauts de page horizontaux de la feuille de calcul

Collection (objet)	Description
VPageBreaks (VPageBreaks)	Les sauts de page verticaux de la feuille de calcul
Hyperlinks (Hyperlink)	L'ensemble des liens hypertextes de la feuille de calcul
Scenarios (Scenario)	Les scénarios de la feuille de calcul
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 hypertextes 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
Workbooks.Add("MonModele.xlt")
'-----
'affectation du nom MaPlage à la plage de cellules A1:C20 de la feuille MaFeuille
ActiveWorkbook.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("Feuill1").Range("C6:E10"), PlotBy:=xlColumns
'définition de l'emplacement du graphique actif
ActiveChart.Location Where:=xlLocationAsObject, Name:= "Feuill1 "
'-----
'sélection de la feuille Feuill1 du classeur actif
Sheets("Feuill1").Select
'affectation du nom Graphique à la feuille Feuill1
Sheets("Feuill1").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!"
'-----
'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

```


Premières macros

L'enregistrement de macros constitue certainement le meilleur apprentissage de Visual Basic pour Applications. Les commandes de l'application hôte accessibles par les onglets 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. Ensuite, vous répétez autant de fois que vous le souhaitez la série d'instructions ainsi mémorisée en exécutant simplement la macro. Vous visualisez le *codage* de la macro dans la fenêtre Code de Visual Basic Editor. Vous découvrez ainsi la structure et la syntaxe des programmes VBA par la pratique.

Définition

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 cette méthode est relativement souple et qu'il existe plusieurs possibilités, plus ou moins efficaces et plus ou moins rapides, pour créer une macro. 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 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 composée de seulement deux commandes, mais elle aidera à découvrir comment 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, 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.

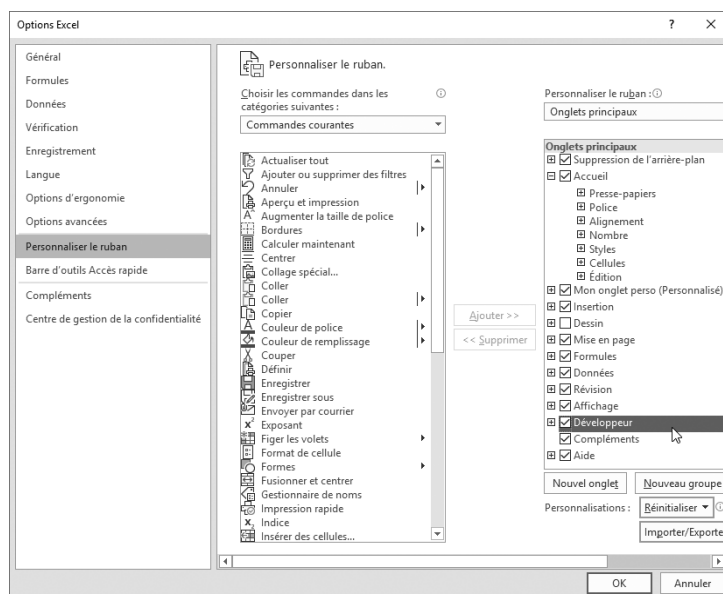


Figure 2-1 – Activez l'onglet Développeur pour accéder aux fonctions de programmation du logiciel.

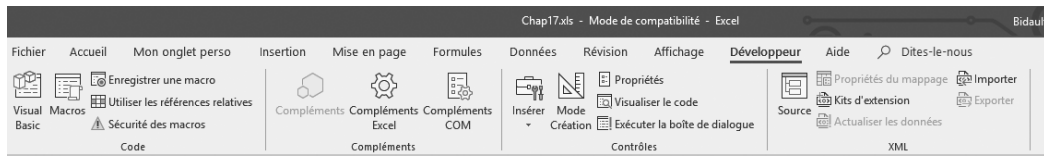


Figure 2-2 – L'onglet Développeur est maintenant accessible sur le ruban.

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.

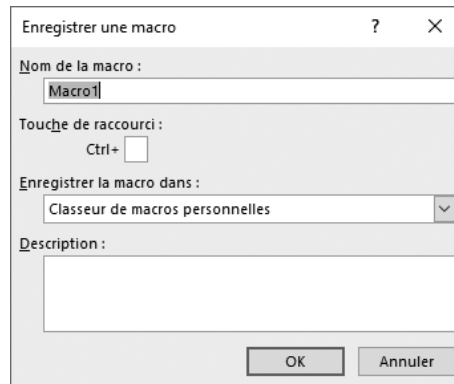


Figure 2-3 – La boîte de dialogue Enregistrer une macro.

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

Conseil

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-leur un nom représentatif et entrez une rapide description de la fonction de chacune 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.
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.

Attention

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 informé. Veillez donc à ne pas attribuer à votre macro un raccourci clavier déjà utilisé par Excel, particulièrement si d'autres utilisateurs sont amenés à utiliser vos macros.

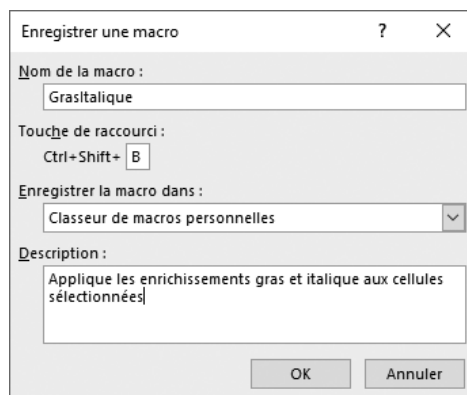


Figure 2-4 – La boîte de dialogue Enregistrer une macro complétée.

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.

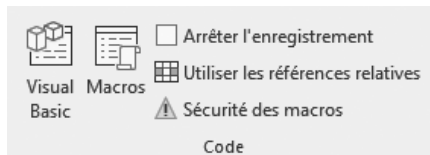


Figure 2-5 – La commande Enregistrer une macro devient Arrêter l'enregistrement.