Programmation OpenOffice.org 2

Couvre les nouveautés de la version 2.0

Macros et API

Bernard Marcelly
Laurent Godard



EYROLLES

Programmation OpenOffice.org 2

Macros OOoBASIC et API

Laurent Godard

Bernard Marcelly

EYROLLES

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

Avec la contribution de Laurent Godard et de Jean-Marie Thomas.



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

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20, rue des Grands-Augustins, 75006 Paris.
© Groupe Eyrolles, 2006, ISBN : 2-212-11763-9

Avant-propos

Rappelons en guise de préambule qu'OpenOffice.org, suite bureautique libre et gratuite, est constituée des modules habituels de traitement de texte, de tableur, de présentation, ainsi que de dessin, d'édition de formules mathématiques et de base de données. Tournant aussi bien sous Windows et Linux que sous Mac OS X, elle peut être utilisée en lieu et place de la suite Microsoft Office, particulièrement dans sa version 2.0 qui offre une compatibilité accrue.

> **À NOTER** **OpenOffice.org et StarOffice**
>
> OpenOffice.org est une base à partir de laquelle diverses variantes ont été dérivées. La plus connue est StarOffice, commercialisée par Sun Microsystems, dont les versions 7 et 8 sont respectivement au niveau des versions 1.1 et 2.0 d'OpenOffice.org. Le contenu de ce livre s'applique généralement à toutes ces variantes.

À qui s'adresse ce livre ?

Vous êtes un utilisateur de la suite OpenOffice.org (ou StarOffice 7 et 8) et vous connaissez bien ses nombreuses possibilités. Cependant, dans certains cas, vous souhaitez simplifier des manipulations répétitives. Le langage de macros Basic d'OpenOffice.org, intégré dans la suite, peut répondre à votre besoin. Il est conçu pour être simple d'emploi, tout en étant puissant.

Avec ce livre, vous apprendrez à programmer la suite OpenOffice.org, par exemple pour ajouter une fonctionnalité personnelle déclenchée par un bouton sur une barre d'outils ou par un raccourci. Vous pourrez même automatiser des traitements, par exemple effectuer

des modifications dans toute une série de documents Writer, ou lire une base de données pour en extraire des informations et les insérer dans un document Writer.

Ce livre est-il accessible à un débutant en programmation ? Les connaissances de base de la programmation sont exposées dans les premiers chapitres. Nous avons aussi inclus des conseils et des bonnes pratiques, qui vous éviteront bien des déboires. À chaque étape, nous avons choisi des exemples volontairement simples, chacun focalisé sur le point à expliquer. Évidemment, le chemin sera plus ardu et plus long si vous en êtes à vos tout débuts : avancez très progressivement, en écrivant de nombreux petits programmes pour vous approprier les concepts de base. Ne craignez pas les erreurs, elles sont source de connaissances. Progressivement, vos programmes s'enrichiront et deviendront toujours plus utiles, à votre grande satisfaction.

Si vous avez l'expérience de la programmation, la première partie vous semblera facile. Méfiez-vous cependant des analogies avec d'autres langages, car chacun a ses particularités. Si vous connaissez la programmation orientée objet, vous comprendrez plus facilement les principes de l'API OpenOffice.org, qui sera utilisée à partir de la troisième partie ; mais ce n'est pas indispensable.

Si vous êtes dans un service informatique chargé d'automatiser des processus utilisant la suite OpenOffice.org ou StarOffice, ou de migrer des applications écrites pour la suite MS-Office, cet ouvrage vous économisera de très nombreuses heures de recherche et de tâtonnements et accélérera la phase d'apprentissage. Comme la mémoire humaine a ses limites, vous souhaiterez garder ce livre toujours à portée de main. Il est cependant probable que vous rencontrerez des besoins non décrits ici, mais la base de connaissances acquises vous facilitera l'étude de l'API OpenOffice.org.

PRÉCAUTION

Pour tirer parti de ce livre, il est recommandé de bien connaître les possibilités d'OpenOffice.org qui s'offrent au niveau de son interface utilisateur, dans les domaines sur lesquels vous souhaitez intervenir par macro. En effet, autant éviter un développement si quelques manipulations résolvent votre problème ou le simplifient. Savez-vous bien utiliser les styles de paragraphe, de caractère, de page ? les modèles de document ? la recherche générique ? le copier/coller avec ou sans formatage ? Savez-vous ce qu'est un signet ? N'hésitez pas à lire ces excellents livres donnant toutes les astuces pour être productif sous OpenOffice.org.

- *OpenOffice.org 1.1 efficace* de Sophie Gautier, Frédéric Labbe, Christian Hardy, Michel Pinquier, Eyrolles 2003 (v2.0 à paraître prochainement).
- OpenOffice.org 2 Writer de Sophie Gautier, Guy Veyssière et Gaël Thomas, Eyrolles 2005.
- OpenOffice.org 2 Calc de Sophie Gautier et Jean-Marie Thomas, Eyrolles 2005.

Contenu de l'ouvrage

Vous trouverez une description complète et précise du Basic OpenOffice.org, rédigée pour être compréhensible pour un débutant en programmation, tout en apportant des informations indispensables au programmeur expérimenté.

Vous apprendrez comment utiliser facilement l'interface de programmation d'application (API) pour lire, écrire, modifier les documents OpenOffice.org, accéder aux bases de données, et dialoguer avec l'utilisateur. L'API d'OpenOffice.org est extrêmement riche et parfois complexe. Il nous a fallu privilégier les sujets les plus courants et les vérifier chacun par des macros. Si vous ne trouvez pas réponse à une question dans cet ouvrage, c'est peut-être qu'il s'agit d'un cas rare et particulièrement difficile à réaliser.

L'API est une interface indépendante du langage de programmation. À cet égard, les descriptions des fonctionnalités sont valides pour d'autres environnements. Un développeur d'applications pourra facilement transposer à un autre langage les exemples Basic donnés dans cet ouvrage. Nous indiquons à l'annexe A les particularités de Basic dans l'utilisation de l'API.

Notre souci a été d'être clair et progressif, sans trop entrer dans des considérations théoriques. Nous avons pour cela créé des centaines d'exemples complets de macros Basic, avec des documents spécialement configurés pour chaque essai. Tous les exemples ont été testés (initialement sur la version 1.1 d'OpenOffice.org et sur une pré-version 2.0 pour ceux qui lui sont spécifiques). Ils sont mis à votre disposition en téléchargement libre sur le site www.editions-eyrolles.com – cela vous épargnera l'effort de frappe et les erreurs de saisie. Ces exemples peuvent servir de modèles pour créer rapidement de nouvelles macros, en quelques copier/coller.

Prenez votre temps en lisant les explications et les exemples : chaque phrase est importante. Nulle prétention littéraire pour ce texte technique. N'hésitez pas à relire des passages que vous pensez connaître.

La source de documentation étant presque exclusivement en anglais, nous avons choisi d'aider le lecteur peu familier de cette langue en traduisant les termes importants et en utilisant des noms de variables en français. À l'usage, le fait d'employer des noms français facilite beaucoup l'assimilation, même si on lit couramment l'anglo-américain.

Ce livre n'est pas une simple traduction de documents anglais, ni une collection d'astuces récoltées sur les forums Internet. Il est une synthèse de connaissances et présente de façon directement utile beaucoup d'informations peu connues, mal documentées ou non documentées. Nous signalons notamment des anomalies de fonctionnement, des limitations, ou des erreurs de documentation afin de vous éviter les difficultés que nous avons rencontrées.

Nous mettons aussi à votre disposition en téléchargement un grand nombre de macros réutilisables qui résolvent les problèmes courants. Leur liste est donnée à l'annexe B.

La **première partie** vous montre pourquoi le langage OpenOffice.org Basic (OOoBasic pour les intimes) est utile, et ce qu'on peut en attendre, c'est-à-dire plus que ne le supposent certains programmeurs chevronnés, habitués aux « vrais langages ».

Nous vous expliquons comment utiliser l'enregistreur de macros, apparu avec la version 1.1, et pourquoi son utilité est finalement assez limitée. Vous faites connaissance avec l'environnement de développement intégré, et vous l'utilisez pour écrire et exécuter votre première macro. Vous apprenez les diverses manières d'exécuter une macro.

Dans la **deuxième partie,** nous décrivons OOoBasic. Il existe plusieurs langages appelés Basic. Ils se ressemblent, mais chacun a ses particularités. Même si vous connaissez déjà un Basic, par exemple Visual Basic ™ qui lui est proche, parcourez les chapitres de cette partie. En cas de problème d'exécution, relisez-la, elle contient bien des détails importants et pas toujours indiqués dans la documentation officielle. Nous vous donnons aussi quelques règles de bonne programmation, non pas pour vous enlever le plaisir de programmer, mais au contraire pour vous éviter échecs et découragements.

À partir de la **troisième partie**, vous apprenez à écrire ou modifier des documents OpenOffice.org : Writer, Calc, Draw, etc. Ceci nécessite d'utiliser l'API, mais nous évitons toute théorie en nous concentrant sur les solutions à des besoins réels. OpenOffice.org réutilise des concepts généraux dans chaque type de documents, mais avec des variations propres à chacun. Nous avons regroupé les principes communs dans le chapitre Accéder aux documents et les aspects spécifiques dans les chapitres suivants. Vous noterez cependant parfois des redondances apparentes, qui sont justifiées par des différences parfois subtiles. Les chapitres les plus importants sont évidemment ceux consacrés aux documents Writer, Calc et Draw. Ils sont assez indépendants, quoique nous vous renvoyions parfois à un autre chapitre (souvent celui consacré au module de traitement de texte Writer) où tel concept a été détaillé. Dans chacun de ces trois chapitres, il n'est nul besoin d'effectuer une lecture complète : après avoir acquis les notions de base, utilisez ensuite le livre comme une référence, et n'approfondissez que les sujets qui vous sont utiles.

Le chapitre concernant les objets insérés dans un document décrit des fonctions présentes dans différents types de documents. Ici l'approche consiste à montrer la méthode pour un premier type de document, puis l'équivalent sur les autres documents.

La **quatrième partie** va au-delà des manipulations de documents pour vous permettre de construire des applications élaborées. Vous y apprenez à afficher des dialogues tout à fait semblables à ceux des applications classiques, à lire et écrire sur des bases de données, et à améliorer vos formulaires avec des macros. Nous vous signalons enfin quelques méthodes sophistiquées, sans prétendre bien sûr à l'exhaustivité.

Enfin nous expliquons en annexe ce qu'est l'API et comment en obtenir des informations pour aller encore plus loin. Nous exposons une liste de macros utilitaires souvent citées au cours de l'ouvrage puis nous signalons divers sites Internet : soit des forums où vous pouvez chercher de l'aide, soit des sites fournissant des exemples de macros, des documents explicatifs, des outils.

ASPECTS JURIDIQUES

Les descriptions, les exemples et les divers fichiers disponibles en téléchargement sont fournis comme potentiellement utiles, mais *sans aucune garantie*, ni explicite ni implicite, y compris les garanties de commercialisation ou d'adaptation dans un but spécifique. Les exemples sont fournis dans un but d'explication et leurs principes sont librement réutilisables.

Certaines routines utilitaires sont soumises à la licence LGPL, décrite sur le site http://www.gnu.org/copyleft/lesser.html. Une traduction non-officielle de la licence LGPL est disponible sur le site http://www.linux-france.org/article/these/licence/lgpl/lgpl_monoblock.html. Brièvement, cela signifie que vous pouvez utiliser ces programmes à condition de maintenir les indications de licence et d'auteur initial.

Ce livre vous fera gagner un temps considérable dans l'apprentissage de la programmation OpenOffice.org. La réalisation d'applications complexes nécessitera parfois d'aller plus loin encore dans l'étude de l'API – disponible seulement en anglais. Mais nous vous faciliterons grandement la tâche en vous donnant les éléments pour la comprendre rapidement.

Commentaires à la deuxième édition

Avec cette nouvelle édition, nous avons ajouté la description des plus importantes modifications apportées par la version 2.0 d'OpenOffice.org à partir des pré-versions disponibles. Les chapitres 2 et 3 ont été très remaniés suite aux évolutions de l'interface utilisateur de la version 2.0. Bien d'autres chapitres ont été mis à jour : correction des erreurs de l'édition originale, particularités apparues avec la version 2.0, amélioration de certains codages, ajout de nouveaux sujets que nous n'avions pas traités précédemment.

Il est possible que dans les versions ultérieures (2.0.1 notamment) certains comportements ou certains aspects de l'interface évoluent par rapport à notre version de travail.

Documents disponibles en téléchargement

Le fichier `.zip` librement téléchargeable sur www.editions-eyrolles.com se décompacte dans un répertoire `MacrosLivre` comprenant autant de sous-répertoires que de chapitres donnant des exemples de macros. Les macros d'un chapitre se trouvent dans des documents Open-

Office.org, ainsi que les fichiers associés éventuels. La référence du fichier à chercher est indiquée en première ligne de chaque macro reproduite dans cet ouvrage.

Avant exécution, il vous faudra copier les fichiers nécessaires dans un répertoire de travail de votre ordinateur, là où, justement, leur exécution est autorisée.

Remerciements

Nous remercions tous les intervenants français et étrangers dans les divers forums OpenOffice.org, soit qu'ils aient apporté leur pierre à la compréhension du produit, soit qu'ils nous aient incités à approfondir tel ou tel domaine grâce à la grande diversité de leurs questions.

Nous remercions tous ceux qui ont bien voulu publier leurs macros, car leur lecture est une source d'information inestimable autant pour les débutants que pour les programmeurs confirmés.

Enfin, merci à Sun MicroSystems pour avoir mis en Open Source la suite OpenOffice.org, donnant ainsi à quiconque la possibilité d'étudier la structure d'un logiciel moderne et de très grande envergure.

Table des matières

PREMIÈRE PARTIE
Introduction aux macros OpenOffice.org 1

CHAPITRE 1
Introduction 3
De l'automatisation d'OOo à l'application d'entreprise 3
Des macros pour les utilisateurs d'OpenOffice.org 4
Des applications à part entière pour l'entreprise 6
Premier aperçu du langage OpenOffice.org Basic 8
OOoBasic, langage de script d'OpenOffice.org 8
OOoBasic et l'API d'OpenOffice.org 9
OOoBasic et VBA 9
Conclusion 10

CHAPITRE 2
Enregistrer et exécuter une macro 11
Les macros et la sécurité 11
Sécurité des macros sur OOo 1.0 et 1.1 11
Sécurité des macros sur OOo 2.0 12
Les différents niveaux de sécurité 12
Les sources de confiance 15
Les signatures numériques 16
Enregistrer une macro 17
Comment enregistrer une macro ? 17
Déclencher facilement une macro 18
Déclencher une macro avec OOo 1.0 ou 1.1 18
Exécuter une macro avec le menu Outils 18
Exécuter une macro depuis un raccourci clavier 18
Exécuter une macro avec un bouton de barre d'outils 19

Exécuter une macro par une entrée de menu 20
Exécuter une macro sur un événement 21
Déclencher une macro avec OOo 2.0 21
Exécuter une macro avec le menu Outils 21
Exécuter une macro depuis un raccourci clavier 23
Exécuter une macro avec un bouton de barre d'outils 24
Exécuter une macro par une entrée de menu 25
Exécuter une macro sur un événement 26
Exécuter une macro en ligne de commande 28
Sous Windows 28
Lancer une macro du conteneur soffice 28
Arguments d'appels de la macro 30
Lancer une macro d'un document OpenOffice.org 30
Sous Linux 31
Lancer un script autre que Basic 32
Conclusion 32

CHAPITRE 3
Créer et gérer les macros depuis l'EDI 33
Premiers pas dans l'environnement de développement Basic 33
L'environnement de développement sous OOo 1.0 et 1.1 34
L'environnement de développement sous OOo 2.0 36
Informations complémentaires 40
Gérer les bibliothèques de macros Basic 41
Renommer une bibliothèque 41
Ajouter une bibliothèque 42
Copier une bibliothèque 43
Protéger une bibliothèque par mot de passe 43
Déplacer un module d'une bibliothèque à une autre 44
Créer, supprimer un module ou une boîte de dialogue 44
Avec OOo 1.0 ou 1.1 44
Avec OOo 2.0 45
Gérer les macros Basic 45
La fenêtre d'édition de macros Basic 46
La coloration syntaxique 47
Ma première macro Basic 47
Exécuter une macro depuis l'éditeur Basic 48
Aides à la mise au point 49
Voir le contenu d'une variable 49

Poser des points d'arrêt 51
Vérifier la syntaxe 51
Modules et macros 51
Autres fonctionnalités de l'éditeur 52
L'environnement de développement des autres langages 52
JavaScript 53
BeanShell 54
Python 55
Java compilé 56
Conclusion 56

DEUXIÈME PARTIE
Le langage OOoBasic 57

CHAPITRE 4
Introduction à la programmation Basic 59
Les éléments de base 59
Les éléments non interprétés 60
Les espaces et tabulations 60
Les commentaires 61
Longueur des lignes 61
Instructions Basic 62
Noms des variables et des routines 62
Syntaxe élémentaire 63
Exécution Basic 64
Recommandations de programmation 64
Le choix des mots 64
La forme : commentaires, espaces 65
Modularité 66
Simplicité 66
Robustesse 66
Relectures 67
Conclusion 67

CHAPITRE 5
Variables et tableaux de variables 69
Déclarer des variables 69
La déclaration Explicit 70

La déclaration Dim 71
Valeur initiale des variables 72
Portée des variables 72
Variable commune à un module 72
Variable commune à une bibliothèque 73
Variable commune à plusieurs bibliothèques 74
Les chaînes de caractères 75
Les variables numériques 76
Les entiers 76
Les nombres réels 77
Les inconvénients du calcul en flottant 78
Les variables monétaires 79
Ordre d'évaluation des opérateurs numériques 81
Les booléens 81
Les variables booléennes 81
Les opérateurs booléens 82
Les calculs booléens sur des nombres 84
Les opérateurs de comparaison 84
Calculs booléens 85
Ordre d'évaluation des opérateurs booléens et de comparaison 85
Les variables de date 86
Les objets 87
Le type Variant 87
La valeur Empty 88
La valeur Null 89
Comment connaître le type d'un Variant ? 89
Les constantes 91
Les tableaux 92
Les tableaux unidimensionnels (vecteurs) 92
Les tableaux multidimensionnels 93
Redimensionner un tableau 94
Connaître les limites d'index 96
Les affectations entre tableaux 97
Les Variant et les tableaux 98
Les tableaux de Variant 98
La fonction Array() 99
Les tableaux irréguliers 99
Diverses fonctionnalités concernant les variables Basic 100
Les caractères de déclaration de type 100

Les caractères de type pour variables non déclarées 100
DimArray() 101
Option Base 102
Conclusion 102

CHAPITRE 6
Conditions, boucles et branchements 103
If Then Else 103
Select Case 106
IIf 108
Choose 109
Switch 110
For Next 111
While Wend 113
Do Loop 114
GoTo, On Goto 115
Conclusion 117

CHAPITRE 7
Les sous-programmes 119
Sub : le sous-programme 119
Appeler un sous-programme 120
Les paramètres de sous-programme 121
Paramètres optionnels 122
Transmission par référence ou par valeur 124
Transmettre un tableau dans un paramètre 125
Portée des variables d'un sous-programme 125
Les variables statiques 126
Fin prématurée d'un sous-programme 127
Gosub : le sous-programme interne 128
Function : le sous-programme fonction 130
Fin prématurée d'une fonction 132
Sous-programmes et bibliothèques 132
Configurations d'appels 133
Charger une bibliothèque 133
Macro initiale dans une bibliothèque de soffice 134
Macro initiale dans un document ouvert 135
Charger une bibliothèque de soffice au démarrage 135
Conclusion 136

CHAPITRE 8

Les principales instructions de traitement..... 137

Les chaînes de caractères 138
Longueur d'une chaîne 138
Comparer deux chaînes de caractères 138
Rechercher une chaîne dans une autre chaîne 138
Le couteau à découper : Mid 139
Supprimer des caractères à gauche et à droite 139
Découper, recoller une chaîne 140
Créer une chaîne de caractères 140
Formater une valeur numérique en chaîne de caractères 141
Les fonctions numériques 141
Signe 141
Fonctions trigonométriques 141
Autres fonctions mathématiques 142
Nombre aléatoire 142
Les fonctions de date et heure 142
Les fonctions de conversion 143
Les conversions automatiques de type 143
Type Boolean vers type numérique 143
Type Boolean vers type String 143
Type numérique vers type Boolean 143
Type numérique vers type String 143
Type String vers type numérique 144
Type String vers type Boolean 144
L'ambiguïté de l'opérateur + ... 144
Les conversions explicites 145
Chaîne de caractères vers nombre 145
Type numérique vers chaîne de caractères 145
Conversions vers un type numérique 146
Conversion d'un nombre réel vers un nombre entier 146
Valeur absolue d'un nombre 147
Changement de casse 147
Conversions de date et heure 148
Fonctions renvoyant une date-heure interne 148
Fonctions prenant pour argument une date-heure interne 148
Conversion vers une valeur booléenne 149

Test de contenu de variable 149
Interface utilisateur : écran, clavier 149
MsgBox 149
Print 151
InputBox 152
Codage des couleurs 152
Traitement des fichiers 153
Syntaxe des adresses de fichiers 153
Gestion de fichiers 154
Explorer un répertoire 154
Lire, modifier des attributs de fichier ou répertoire 157
Écrire, lire un fichier 157
Fichier texte 158
Fichier texte pour sauver des données 159
Fichier binaire à accès direct 160
Fichier binaire pur 162
Autres instructions Basic pour fichiers ouverts 162
Fonctions système 162
Lancer un programme externe 163
Conclusion 163

CHAPITRE 9
Le traitement des erreurs d'exécution 165
Le mécanisme d'interception d'erreur 165
Un exemple typique 166
Conséquence d'une instruction non exécutée 167
Reprise du traitement à un autre endroit 167
L'instruction Resume est indispensable 168
Ignorer les erreurs 168
Informations sur l'erreur 169
Portée d'un traitement d'erreur 173
Le traitement d'erreurs pour simplifier le codage 175
Déclencher une erreur 175
Conclusion 175

TROISIÈME PARTIE
Manipuler les documents OpenOffice.org 177

CHAPITRE 10
Les documents OpenOffice.org 179
Accéder au document 180
Accéder au document en cours 180
Accéder à un autre document 181
Propriétés d'ouverture de document 183
Demander le mot de passe du document 185
Éditer un document modèle 186
Créer un nouveau document 186
Créer un nouveau document conforme à un modèle 186
Sauver un document 187
Avant de sauver... 187
Réaliser la sauvegarde 188
Enregistrer une copie 189
Fermer le document 189
Exemples récapitulatifs 190
Les filtres d'import/export 193
Principes de base 194
Exporter en PDF, Flash, HTML 195
Importer, exporter au format CSV 197
Les paramètres du filtre CSV 198
Importer, exporter du texte pur 200
Imprimer un document 200
L'objet Printer 201
Changer la configuration d'impression 203
Lancer l'impression 204
Les informations du document 205
Informations spécifiques à un document Writer ou HTML 208
Configuration d'affichage d'un document 209
Configuration d'un document 210
Conclusion 211

CHAPITRE 11
Les documents Writer 213
Imprimer 213

L'objet Text 214
Le curseur d'écriture 215
Déplacer le curseur d'écriture 215
Autres initialisations d'un curseur 217
Lire une zone de texte 218
Insérer du texte 219
Avec la propriété String 219
Avec la méthode insertString 220
Insérer des caractères spéciaux 221
Insérer un saut de page ou de colonne 222
Insérer le texte d'un autre document 223
Supprimer des paragraphes 223
Supprimer une marque de paragraphe 224
Supprimer tout un paragraphe 224
Appliquer un formatage 225
Appliquer un style à un paragraphe 225
Appliquer un style à un ou plusieurs caractères 226
Formatage local des caractères 227
La « graisse » 227
Italique 228
Soulignement 228
Accentuation 229
Relief 229
Changement de casse 230
Exposant et indice 230
Couleurs 230
Autres propriétés de caractère 231
Supprimer tout formatage de caractère 232
Curseur visible et zone sélectionnée par l'utilisateur 232
Obtenir le curseur visible 232
Zone sélectionnée par l'utilisateur 233
Modifier le contenu de la zone 233
Définir un curseur d'écriture sur la zone sélectionnée 233
Obtenir l'objet texte de la sélection 234
Où se trouve le curseur ? 236
Explorer la zone sélectionnée par l'utilisateur 236
Sélectionner de manière visible une zone de texte 237
Déplacer le curseur visible 238
Définir des positions de tabulations 239

Rechercher, remplacer 241
Le descripteur de recherche 241
Limiter le champ de la recherche 242
Rechercher pour remplacer 245
Tout remplacer 246
Rechercher des paragraphes 248
Les tableaux 249
Insérer un tableau 249
Insérer plusieurs tableaux 250
Trouver un tableau existant 250
Supprimer un tableau 251
Propriétés de tableau 252
Bordures de tableau 253
Ombre de tableau 255
Largeur du tableau 256
Lignes 258
Colonnes 259
Se déplacer dans un tableau 261
Le curseur de cellule 261
Se déplacer sans curseur de cellule 263
Zone sélectionnée par l'utilisateur 264
Les cellules 265
Bordures de cellule 265
Écrire un texte dans la cellule 266
Formules et valeurs numériques 267
Trier un tableau 267
Tableaux irréguliers 271
Les cadres 273
Insérer un cadre 273
Insérer plusieurs cadres 274
Trouver un cadre 274
Supprimer un cadre 275
Dimensionner un cadre 276
Positionner le cadre 276
Les différents ancrages de cadre 276
Positionnement horizontal 277
Positionnement vertical 278
Adaptation du texte 281
Autres propriétés de cadre 281

Écrire du texte dans un cadre 282
Les sections 283
Créer une section, écrire dedans 283
Naviguer dans les sections 284
Propriétés des sections 286
Les styles 287
Trouver les styles 287
Récupérer ou supprimer un style existant 288
Créer un nouveau style 290
Les propriétés de style 290
Style de paragraphe 290
Style de caractère 292
Style de page 292
Comment utiliser les styles de page 294
Style de cadre, style de numérotation 296
Les en-têtes et pieds de page 296
Les champs de texte 298
Variables de champ utilisateur 299
Champs d'utilisateur 299
Notes 302
Textes et paragraphes conditionnels 304
Les signets et renvois 306
Utiliser un signet existant 306
Insérer un signet 308
Trouver les signets 309
Liens hypertextes 309
Lien externe 310
Lien vers un autre document 310
Lien interne au document 310
Lien vers un endroit dans un autre document 311
Configuration d'affichage du document 311
Conclusion 313

CHAPITRE 12
Les documents Calc 315
Lecture et manipulation de feuilles 315
Accéder aux feuilles existantes 315
Ajouter une nouvelle feuille 317
Supprimer une feuille 318

Dupliquer une feuille 318
Déplacer une feuille dans le classeur 319
La feuille visible par l'utilisateur 319
Protéger une feuille 320
Cellules et zones de cellules 321
Obtenir une cellule 321
Obtenir une zone de cellules 322
Obtenir les coordonnées d'une cellule 323
Obtenir les coordonnées d'une zone de cellule 323
Les zones nommées 324
Les sélections visuelles 325
Sélection faite par l'utilisateur 325
Afficher une zone sélectionnée 327
Zone visible dans la feuille 327
Figer des lignes ou colonnes 327
Première ligne et première colonne affichées 328
Propriétés globales de la cellule 328
Format de nombre 329
Protection de la cellule 330
Alignement horizontal 330
Alignement vertical 330
Orientation du contenu 331
Bordures de la cellule 331
Bordures d'un tableau 332
Ombre de la cellule 334
Note (annotation) de cellule 335
Lignes et colonnes 337
Les lignes 337
Les colonnes 338
Lire et écrire dans une cellule 339
Les différents contenus d'une cellule 339
Le curseur d'écriture dans la cellule 340
Déplacer le curseur d'écriture 341
Créer un curseur à partir d'un autre curseur 342
Lire un texte dans une cellule 343
Insérer un texte dans une cellule 343
Insérer des caractères spéciaux 344
Formater la cellule 345
Imposer son style 345

Formatage local de la cellule 345
Les formats de nombre 347
Méthodes pratiques 351
Activer le calcul des formules 352
Effacer une zone de cellules 352
Énumérer les cellules d'une zone 353
Fonctions mathématiques sur une zone de cellules 354
Lire et écrire les données d'un tableau 355
Coller uniquement les valeurs et non le format 357
Déplacer ou recopier des cellules avec références 357
Déplacer une zone de cellules 357
Recopier une zone de cellules 358
Recopier une cellule dans une zone 358
Recopier une formule dans une zone 360
Rechercher, remplacer 361
Le descripteur de recherche 362
Limiter le champ de la recherche 363
Rechercher pour remplacer 365
Tout remplacer 365
Trier un tableau 366
Utiliser une fonction de Calc 370
Créer une nouvelle fonction pour Calc 372
Insérer un lien hypertexte 374
Configuration d'affichage du document 374
Imprimer 375
Zones d'impression 375
Répéter les en-têtes 376
Les diagrammes 377
Obtenir un diagramme existant 377
Les propriétés d'un diagramme 379
Les styles 380
Trouver les styles 380
Récupérer ou supprimer un style existant 381
Créer un nouveau style 383
Les propriétés de style 383
Style de cellule 383
Style de page 383
Comment utiliser les styles de page 386
Les en-têtes et pieds de page 387

Configuration de Calc 389
Conclusion 392

CHAPITRE 13
Les documents Draw et Impress **393**
Les pages de dessin 394
Accéder aux pages existantes 394
Renommer une page 395
Ajouter une nouvelle page 396
Supprimer une page 397
Dupliquer une page 398
Déplacer une page dans la liste des pages 398
La page visible par l'utilisateur 398
Les arrière-plans 399
Accéder aux arrière-plans existants 399
Renommer un arrière-plan 400
Ajouter un arrière-plan 401
Supprimer un arrière-plan 402
Dupliquer un arrière-plan 402
Déplacer un arrière-plan dans la liste des arrière-plans 402
Affecter un arrière-plan à une page 402
Les couches 403
Accéder aux couches existantes 403
Les couches et l'interface utilisateur 404
Renommer une couche 405
Ajouter une couche 405
Supprimer une couche 406
Dupliquer une couche 407
Déplacer une couche dans la liste des couches 407
Les propriétés d'une couche 407
Changement du mode d'affichage 407
Les propriétés d'une page de dessin 408
Dessiner une forme 409
Trouver une forme existante 410
Trouver une forme nommée 410
Trouver les formes sélectionnées par l'utilisateur 411
Sélectionner visiblement une forme 412
Lister les formes d'une page 412
Supprimer une forme 413

Propriétés des formes 414
Type de la forme 414
Position et taille de la forme 414
Les liens entre forme et couche 414
Le contour de la forme 416
Les lignes tiretées 416
Le fond et la forme 418
Les styles de remplissage du fond 418
Couleur de fond 419
Un peu de transparence 422
Fond hachuré 424
Fond à motif bitmap 426
L'ombre d'une forme 427
Angle de rotation de la forme 428
Cisaillement de la forme 428
Écrire un texte dans une forme 428
Gestion globale du texte 428
Position du texte dans la forme 430
Texte animé 431
Utilisation d'un curseur d'écriture 432
Les différentes formes 434
Le rectangle, le carré 434
L'ellipse, le cercle 434
Le texte 435
La ligne simple 436
La ligne brisée 437
Le polygone 438
Le poly-polygone 439
Le connecteur 440
L'étiquette 442
La ligne de cote 443
Les formes de Bézier 445
Collages 448
Les points de colle 448
Ajouter un point de colle 451
Supprimer un point de colle 451
Relier deux formes par un connecteur 452
Manipuler plusieurs formes 453
L'ordre Z 453

Grouper des formes 453
Combiner plusieurs formes 454
Connecter plusieurs formes 454
Exporter une forme 455
Les styles 459
Trouver les styles 459
Modifier un style existant 460
Supprimer un style existant 460
Créer un nouveau style 461
Imprimer 461
Configuration d'impression 461
Configuration du document 462
Spécificités d'Impress par rapport à Draw 464
Couches de dessin (version 2.0.0) 464
Exécuter une macro pendant une présentation 464
La page de notes 464
La page prospectus 465
Les styles Impress 465
Configuration de document 466
Configuration d'impression 466
Accès aux mécanismes d'Impress 467
Conclusion 467

CHAPITRE 14
Les objets insérables dans un document 469
Classification des objets insérables 469
Les formes, ou dessins 470
Les formes dans un document Writer 470
Insérer une forme à la position du curseur 471
Insérer plusieurs formes 472
Retrouver, supprimer une forme 473
Interaction entre la forme et les autres objets 473
Les formes dans un document Calc 474
Insérer plusieurs formes 475
Retrouver, supprimer une forme 475
Les images 476
Les images dans Draw 477
Insérer plusieurs images 478
Retrouver, supprimer une image 478

Les images dans un document Writer 479
Insérer une image à la position du curseur 479
Insérer plusieurs images 480
Retrouver, supprimer une image 481
Les images dans un document Calc 482
Insérer plusieurs images 482
Retrouver, supprimer une image 483
Propriétés des images 484
Intégrer une image dans le document 485
Manipuler les équations 486
Conclusion 488

QUATRIÈME PARTIE
Construire des applications avec OpenOffice.org ..489

CHAPITRE 15
Les boîtes de dialogue 491
Construire une boîte de dialogue avec l'EDI 491
Le panneau de dialogue 492
La fenêtre des propriétés 493
L'onglet Général 493
L'onglet Événements 494
Votre première boîte de dialogue 495
L'étiquette 496
Le Bouton 497
Le bouton Standard 497
Le bouton OK 497
Le bouton Annuler 497
Le bouton Aide 497
Ajuster les éléments du dialogue 497
Tester le dialogue 498
Exécuter le dialogue 499
Accéder aux bibliothèques de dialogues 500
Les principaux champs de saisie 501
La zone de texte 501
Le champ numérique 503
La zone de liste 504
La zone de liste simple 504
La zone de liste à sélection multiple 506

La zone de liste combinée 507
Les cases à cocher 508
Les cases de choix 1 parmi N 509
L'aspect visuel des boîtes de dialogue 511
Cohérence des dimensions 511
La notion de « Focus » 512
Lettre accélératrice 512
Le bouton par défaut 513
Les éléments visuels 513
La zone de groupe 513
Les lignes horizontale et verticale 514
Les champs de saisie spécialisés 514
Le champ de date 514
Le champ horaire 516
Le champ monétaire 517
Le champ masqué 519
Le champ formaté 520
Format Date 520
Format Pourcentage 522
Format Monétaire 523
Format Booléen 524
Format Fractionnaire 525
La sélection de fichiers 526
Le contrôle Image 527
Les barres de défilement 528
La barre de progression 529
Conclusion 532

CHAPITRE 16
Créer des boîtes de dialogue élaborées 533
Modifier dynamiquement un contrôle 533
Le contrôle et son modèle 533
Exemple de modification d'un dialogue 534
Programmation du contrôle zone de liste 537
Mettre le focus sur un contrôle 539
Remettre à « vide » un champ numérique 539
Traiter des événements dans un dialogue 540
Gestionnaire d'événement commun à plusieurs contrôles 541
Les informations fournies par un événement 543

Lors du déclenchement 543
Changement de focus 544
Touche du clavier 544
Souris 546
Texte Modifié 547
Dialogues à pages multiples 547
Dialogues emboîtés 548
Position et dimensions 548
Un dialogue dans un dialogue 548
Les services de dialogues 552
Sélectionner un fichier existant 553
Sélectionner plusieurs fichiers existants 554
Enregistrer un fichier 555
Les différents dialogues de FilePicker 556
Choisir un répertoire 557
Conclusion 558

CHAPITRE 17
Les sources de données 559
Afficher les outils intégrés 560
Le gestionnaire de sources de données 560
L'assistant de création d'états 561
OpenOffice.org 2.0 562
Les sources de données 563
Lister les sources de données 563
Propriétés d'une source de données 565
Créer et supprimer une source de données 566
OpenOffice.org 1.1.x 567
OpenOffice.org 2.0 569
Se connecter à une source de données 570
Demande de mot de passe à l'utilisateur 572
Propriétés d'une connexion 573
Les tables 574
Les Champs 576
Créer une table 578
Les requêtes pré-enregistrées 580
Enregistrer une requête 581
Modifier une requête 582
Ouvrir un rapport (ou état) 582

Accéder aux données avec le langage SQL 584
Exploiter les résultats d'une requête 584
Autres exemples de requêtes d'interrogation 589
Insérer un enregistrement dans une table 589
Avec l'instruction SQL 589
Avec le ResultSet 592
Modifier un enregistrement d'une table 593
Avec l'instruction SQL 593
Avec un ResultSet 593
Effacer des enregistrements 594
En utilisant l'instruction SQL 594
En utilisant le ResultSet 595
Les différentes capacités des ResultSet 595
Les valeurs nulles 596
Les requêtes paramétrées 596
Accéder aux données avec un RowSet 597
Créer un RowSet 598
Tri supplémentaire 600
Filtre supplémentaire 601
Utiliser les requêtes pré-enregistrées 602
Les événements du RowSet 602
Les transactions 605
Ouverture de la base 605
Gérer les transactions 607
Utilisation dans le contexte bureautique 608
Le publipostage 609
Une requête dans Calc 611
Une requête dans une cellule 611
Importer des données 612
Importer les requêtes MS-Access 614
Conclusion 616

CHAPITRE 18
Les formulaires 617
Accéder aux contrôles d'un formulaire 618
Les sous-formulaires 621
Fonctionnalités de base des contrôles 621
Le bouton 622
Le bouton picto, ou bouton-image 622

Les zones de liste 622
La zone de liste simple 622
La zone de liste combinée 623
La case à cocher 623
Le choix 1 parmi N 623
Les champs de saisie 624
Autres contrôles 625
Contrôles et base de données 625
L'objet formulaire 625
Le contrôle Table 625
Connaître la sélection dans le contrôle Table 627
Déplacer la position courante dans le contrôle Table 628
Les contrôles champ de données 629
Zone de liste et base de données 631
Les événements des formulaires 632
Formulaire intelligent 635
Principes communs aux contrôles 635
Imposer le focus sur un contrôle 635
Gestionnaire d'événement commun à plusieurs contrôles 635
Les formulaires intégrés dans un document Base 636
Ouvrir un formulaire d'un document Base 636
Ouvrir un formulaire en mode conception 638
Obtenir les contrôles d'un formulaire intégré 638
Conclusion 640

CHAPITRE 19
Techniques avancées pour le poste de travail 641
Les répertoires d'installation 641
Gérer les fichiers depuis l'API 643
Écrire un fichier binaire 644
Lire un fichier binaire 645
Lecture-écriture d'un fichier binaire 646
Création et décompression d'un fichier ZIP 646
Lancer l'application associée à un document 648
La palette des couleurs 649
Penser à l'utilisateur 650
Geler l'interface utilisateur 650
Empêcher les actions de l'utilisateur 651
État d'avancement du travail 652
Visibilité du document 653

Traitements spécifiques à MS-Windows 653
Accéder à la base de registres de MS-Windows 653
Utiliser l'API de MS-Windows 654
Manipuler les objets COM 655
Envoyer un document par courrier électronique 656
Utiliser un serveur Web 658
Intercepter un événement (listener) 659
Modifier des macros par programmation 663
L'écriture dynamique de macros 663
Modifier les macros d'un autre document 665
Particularités apportées par OOo 2.0 666
Résultat du Dispatcher 666
Appeler un script écrit dans un autre language 667
Conclusion 669

ANNEXES
Outils et ressources 671

ANNEXE A
Comprendre l'API d'OpenOffice.org 673
Qu'est-ce que l'API ? 673
L'API réelle et l'API selon OOoBasic 675
Les fonctions Basic dédiées à l'API 677
Comprendre les messages d'erreur OOoBasic 679
Évitez les erreurs 680
Erreur : variable indéfinie 680
Erreur : variable objet non paramétrée 680
Erreur : utilisation incorrecte d'un objet 682
Erreur : propriété ou méthode introuvable 682
Erreur : la sous-procédure ou procédure fonctionnelle n'est pas définie 682
Erreur : runtime exception 682
Erreur : cannot coerce argument type during corereflection call ! 683
L'API avec d'autres langages de programmation 684
Les autres langages de script de la version 2 684
JavaScript et BeanShell 684
Python 684
Piloter OpenOffice.org avec Microsoft COM 685
La documentation de l'API (Software Development Kit) 687
Comment s'y retrouver ? 688
Conclusion 689

ANNEXE B
Routines utilitaires ... 691
Tableaux de propriétés ... 691
Collections ... 693
Coordonnées de cellules ... 693
Rechercher un objet par son nom ... 695
Rechercher un diagramme par son nom ... 696
Redimensionner une image ... 697
Traduire un nom de style ... 698
Rappel des routines déjà décrites dans les chapitres ... 699
Dialogue ... 699
Tableur et base de données ... 699
Rechercher la forme d'un contrôle de formulaire ... 699
Création et décompression d'un fichier ZIP ... 699
Conversion date-heure vers heure, minute, seconde ... 699
Traitement des chaînes de caractères longues ... 700
Remplacer un motif partout dans une chaîne de caractères ... 701
Trier un tableau de données ... 701
Obtenir des adresses URL de fichiers et répertoires ... 702
Copier-coller avec le presse-papiers ... 702
Conclusion ... 703

ANNEXE C
Ressources disponibles sur Internet ... 705
Diffuser vos macros avec un add-on ... 705
Introspection et documentation avec Xray ... 707
Les bibliothèques de DannyB ... 710
Documents informatifs et exemples ... 711
Sites Web et forums ... 712
En français ... 713
En anglais ... 713
Autres langues ... 714
IssueZilla ... 714
Rechercher un rapport dans IssueZilla ... 714
Rédiger un rapport ... 715
Partager la connaissance ... 716
Conclusion ... 716

Index ... 717

Première partie

Introduction aux macros OpenOffice.org

Les chapitres de cette partie vous montrent pourquoi le langage OpenOffice.org Basic (OOoBasic pour les intimes) est utile, et ce qu'on peut en attendre, c'est-à-dire plus que ne le supposent certains programmeurs chevronnés, habitués aux « vrais langages ».

Vous verrez également comment utiliser l'enregistreur de macros, apparu avec la version 1.1, et pourquoi son utilité est finalement assez limitée. Vous ferez connaissance avec l'environnement de développement intégré, et vous l'utiliserez pour écrire et exécuter votre première macro. Enfin, vous apprendrez les diverses manières d'exécuter une macro.

1

Introduction

Avant d'aborder dans les chapitres suivants l'apprentissage progressif d'OOoBasic, il nous a semblé utile de passer en revue un éventail des besoins auxquels il peut répondre, puis de préciser comment il se situe par rapport à d'autres langages de programmation et dans quel contexte y recourir.

De l'automatisation d'OOo à l'application d'entreprise

Avant de vous lancer dans l'aventure, vous vous demandez peut-être ce qu'on peut bien réaliser d'intéressant avec OOoBasic et l'API d'OpenOffice.org. Eh bien, tout est fonction du besoin. Une « bonne » macro est une macro qui satisfait un besoin, qu'il soit récurrent ou ponctuel. Il n'est pas nécessaire de bâtir un environnement applicatif complet (même si cela est tout à fait possible) et quelques lignes suffisent parfois à rendre des services inestimables au quotidien.

Les macros d'OpenOffice.org permettent d'adapter le logiciel à un besoin spécifique, avec cet avantage indéniable que dans le cas des macros OOoBasic, tout est déjà intégré et prêt à l'utilisation. OOoBasic offre un cadre d'exécution commun pour élaborer des additifs logiciels. Une macro « arrivant » sur un poste est certaine de retrouver ce cadre de travail, et ce quelle que soit la plate-forme utilisée.

ALTERNATIVES **Et les autres langages ?**

La version 2.0 d'OpenOffice.org permet d'utiliser d'autres langages que Basic pour écrire des scripts. Nous aborderons ces langages aux chapitres 3 et 19, ainsi qu'à l'annexe A. OOoBasic reste cependant de loin le langage le plus pratique.

Des macros pour les utilisateurs d'OpenOffice.org

Les utilisations des macros peuvent être multiples. On peut bien sûr intervenir directement sur un document en cours pour reproduire une tâche répétitive ou fastidieuse, mais aussi fédérer plusieurs documents pour des traitements transversaux. Bien des utilisateurs ont leurs propres macros, non publiées, qui leur font gagner du temps dans leurs activités quotidiennes, depuis l'application d'un style de caractère en cliquant sur une simple icône jusqu'à la mise en forme de plusieurs documents à la fois (voir tableau 1-1). Vous remarquerez que les fonctionnalités de certaines macros sont maintenant intégrées dans la version 2.0 d'OpenOffice.org.

Dans le tableau 1-1, nous ne citons que des macros apportant une nouvelle fonctionnalité à l'utilisateur final.

Tableau 1–1 Macros enrichissant OpenOffice.org de nouvelles fonctions

Nom du programme	Utilité	Site Internet
Multipages	Manipule aisément un grand nombre de feuilles dans Calc, ou de pages dans Draw, ou de diapositives dans Impress.	http://fr.openoffice.org/Documentation/Outils/index.html
Multisave	Enregistre simultanément un document aux formats `.doc`, `.pdf` et `.sxw`.	http://fr.openoffice.org/Documentation/Outils/index.html
PickList	Permet de changer le nombre d'entrées dans la liste des fichiers récemment ouverts ou de supprimer cette liste.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Détacher du modèle	Casse le lien entre un document et son modèle.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Casse	Change la casse des lettres d'un texte (c'est-à-dire les met en majuscules ou minuscules).	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Style	Permet d'attacher un raccourci clavier à un style. Obsolète sur OOo 2.0	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Compte Mot	Pour les journalistes, compte les mots dans une sélection de textes. Obsolète sur OOo 2.0	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Transpose	Pour corriger vos fautes de frappe, inverse les deux lettres encadrant la position du curseur.	http://fr.openoffice.org/Documentation/Macros/indexmac.html

Tableau 1–1 Macros enrichissant OpenOffice.org de nouvelles fonctions

Nom du programme	Utilité	Site Internet
Supprimparag	Supprime les fins de paragraphes inutiles dans un document récupéré d'une autre application.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
FusionDocument	Lors d'un publipostage, permet de fusionner les documents dans un seul fichier.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Exposant	Met en exposant toutes les abréviations d'énumération.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
GrilleCalc	Permet d'afficher à volonté la grille délimitant les cellules dans un document Calc.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
Vocabulary	Un traducteur de mots.	http://fr.openoffice.org/Documentation/Outils/index.html
OOoVirg	Installe une icône pour activer le remplacement automatique du point du clavier numérique par la virgule. Obsolète sur OOo 2.0	http://fr.openoffice.org/docs/InstallVirgule1.2.sxw
Pinceau (PincOOo)	Réalise l'équivalent du « pinceau à formater » qui existe sous MS-Word. Obsolète sur OOo 2.0	http://fr.openoffice.org/Documentation/Outils/index.html
TailleFormules	Sert à redimensionner facilement la taille des formules (ou équations) incluses dans un document.	http://fr.openoffice.org/Documentation/Outils/index.html
DMaths	Un ensemble de macros et une barre d'outils pour ceux et celles qui écrivent de nombreuses équations mathématiques sous Writer.	http://www.dmaths.com/
Advanced Data Source Settings GUI	Modifie certains paramètres de sources de données de manière conviviale.	http://ooomacros.org/user.php
Duplicate Shape	Fait une copie d'une forme Draw sélectionnée et la positionne juxtaposée à celle-ci.	http://ooomacros.org/user.php
Extended PDF	Une autre manière de réaliser une version PDF d'un document OpenOffice.org, en incluant les signets, hyperliens, et les informations de document. Obsolète sur OOo 2.0	http://ooomacros.org/user.php
Graph Plotter	Dessine des axes et graphiques dans un document Draw.	http://ooomacros.org/user.php
History Manager	Change la taille de la Picklist et de l'historique des fichiers chargés.	http://ooomacros.org/user.php
OOo Statistics	Ajoute à Calc des fonctionnalités de statistiques équivalentes à Analysis Tool Pack pour Excel.	http://ooomacros.org/user.php
Text to Columns	Permet de séparer en plusieurs colonnes du texte ou des données d'un tableur Calc.	http://ooomacros.org/user.php
CadOOo	Ajoute des fonctionnalités de dessin	http://fr.openoffice.org/Documentation/Outils/cadooo-0.10.2.zip

Tableau 1–1 Macros enrichissant OpenOffice.org de nouvelles fonctions

Nom du programme	Utilité	Site Internet
Ponctuation double	Place un espace insécable avant et après une ponctuation double.	http://fr.openoffice.org/Documentation/Macros/insecable_pctua_double.sxw
ListeValidation	Modèle permettant d'utiliser les listes de validation.	http://fr.openoffice.org/Documentation/Macros/CalcListesValidationMacro2.stc
Tri dans Calc	Macro permettant d'améliorer les fonctions de tri dans Calc.	http://fr.openoffice.org/Documentation/Macros/Macro_Outil_Tri_Calc_0.6.sxw
Ecrire des nombres en lettres	Une nouvelle fonction pour Calc, pratique pour les littéraires ou pour rédiger un chèque !	http://fr.openoffice.org/Documentation/Exemples/MacroNombreEcritEnLettre.sxc

Des applications à part entière pour l'entreprise

Un nombre croissant d'entreprises et d'administrations développent des applications internes basées sur OOoBasic et l'API d'OpenOffice.org. Des outils internes à l'API peuvent notamment permettre d'envisager une utilisation à travers un réseau voire Internet. Là encore, de nombreuses fonctionnalités sont présentes en interne.

Par exemple, si un important fonds documentaire est disponible dans un certain format et qu'il devient nécessaire d'en effectuer une migration pour obtenir une version PDF des documents, les fonctionnalités d'import/export le permettent.

Si des données sont éparpillées dans plusieurs sources et qu'il devient nécessaire de les féderer voire d'en construire des graphes à intervalles donnés, l'accessibilité à l'API de Calc va pouvoir répondre au besoin.

Si un mailing requiert des interventions particulières ou s'il devient nécessaire de récupérer des données dans des documents contenant des champs utilisateurs afin de les consolider, là encore, l'API et les macros peuvent être utilisées.

Enfin, les macros peuvent servir à faire de petits scripts simples complètement déconnectés du contexte bureautique, comme des « moulinettes » sur des fichiers texte.

> La plupart de ces exemples sont tirés du site officiel de la communauté OpenOffice.org francophone. Nous n'y incluons pas les macros servant d'outils pour la programmation en OOoBasic, ni les macros qui servent d'exemples de programmation. Certaines de ces macros constituent de véritables programmes à part entière.
>
> ▸ http://fr.openoffice.org/

Tableau 1–2 Des applications à part entière réalisées en macros OOoBasic

Nom du programme	Utilité	Site Internet
Faire-Part de naissance	Réaliser facilement un petit livret par simple pliage d'une feuille.	http://fr.openoffice.org/Documentation/Outils/index.html
Master	Un jeu type MasterMind.	http://fr.openoffice.org/Documentation/Outils/index.html
Publipostage	Automatise la création de courriers types à travers une base de données.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
RenommerPhotos	Permet de renommer une série de fichiers selon une table de correspondance définie dans une feuille Calc.	http://fr.openoffice.org/Documentation/Macros/indexmac.html
OOoCredit	Calcule le Taux Effectif Global (TEG) d'un crédit.	http://fr.openoffice.org/Documentation/Exemples/indexex.html
FitOO	Pour les scientifiques, un outil de corrélation non-linéaire, très utile pour modéliser une courbe résultant d'un relevé de mesures.	http://fr.openoffice.org/Documentation/Outils/index.html
BatchConv	Pour convertir une liste de fichiers depuis tout format pris en compte à l'importation par OpenOffice.org vers tout format pris en compte pour l'exportation.	http://fr.openoffice.org/Documentation/Outils/index.html
OOoconv	Une autre version de BatchConv, qui fonctionne comme serveur de conversion, accessible sous Internet ou intranet.	http://oooconv.free.fr/
Calc2qif	Sert à convertir des données Calc au format QIF, utilisé par les outils de gestion monétaire comme Money, Quicken, GNUCash.	http://ooomacros.org/user.php
Impress Photo Album Creator	Pour créer un album photo avec Impress.	http://ooomacros.org/user.php
Slide Splitter	À partir d'un document Impress, réalise un fichier par diapositive, au format PDF ou Flash.	http://ooomacros.org/user.php
TablAmortPret	Calcule l'amortissement d'un prêt	http://fr.openoffice.org/Documentation/Outils/TablAmortPret.sxc
Tableau	Tableau d'amortissement linéaire des immobilisations.	http://fr.openoffice.org/Documentation/Outils/Tableau.sxc
GenealOOogie	Lit et analyse un fichier GED, utilisé en généalogie.	http://fr.openoffice.org/Documentation/Outils/GenealOOogie1.1.zip.tgz
OOoWikipedia	Pour se connecter à Wikipedia.	http://fr.openoffice.org/Documentation/Outils/InstallOOoWikipedia1.3.sxw
WikiWiki	Permet un copier/coller entre OOo et un WikiWiki.	http://fr.openoffice.org/Documentation/Macros/WikiWiki_beta07.sxw
Milifred	Un traceur de papier semi-log ou log-log.	http://fr.openoffice.org/Documentation/Outils/milifred2.sxd

Tableau 1–2 Des applications à part entière réalisées en macros OOoBasic

Nom du programme	Utilité	Site Internet
ODevisClicFacture	Un outil pour réaliser devis et facture.	http://fr.openoffice.org/Documentation/Outils/oDevisClicFacture2.3beta79.zip
RibanSir	Outil pour contrôler les RIB, IBAN, SIREN, SIRET, CB.	http://fr.openoffice.org/Documentation/Outils/RISC103.sxw
O_Clic_CP	Recherche et saisie de codes postaux.	http://fr.openoffice.org/Documentation/Outils/O_Clic_CP_1.0.zip

> **VERSION OpenOffice.org 1.1 et OpenOffice.org 2.0**
>
> Les exemples de ce livre ont été testés sur la version 1.1 et, pour certains, sur une pré-version 2.0.0 d'OpenOffice.org. Ils peuvent pour la plupart fonctionner sur une version précédente (1.0.3 ou 1.0.2) mais nous vous conseillons très fortement d'utiliser une version au moins égale à 1.1. En effet, de nombreuses anomalies ont été corrigées à chaque nouvelle version. De nouvelles fonctionnalités, pas toujours apparentes pour l'utilisateur, mais qui ont un impact sur la programmation, ont aussi été introduites.

Premier aperçu du langage OpenOffice.org Basic

> Le langage Basic de la suite OpenOffice.org est parfois appelé en abrégé OOoBasic afin de le distinguer des autres langages Basic, et nous reprendrons ce terme. La société Sun Microsystems préfère le terme StarOffice Basic ou StarBasic, car elle diffuse la suite StarOffice™ qui est basée sur OpenOffice.org.

OOoBasic, langage de script d'OpenOffice.org

Son nom rappelle un des plus anciens langages, Basic, conçu à l'origine pour des débutants en programmation disposant de moyens informatiques parfois très limités. En réalité OOoBasic n'a plus qu'un lointain rapport avec son ancêtre. Bien qu'il reste d'un abord simple, OOoBasic est un langage de programmation moderne permettant de développer des applications sophistiquées. C'est un langage de programmation modulaire : fini les numéros de lignes, les `GOTO` et la programmation « spaghetti » ; les macros sont regroupées en modules et en bibliothèques réutilisables.

OOoBasic est entièrement intégré à OpenOffice.org, et ce dernier comporte un environnement de développement intégré (EDI) servant à écrire les macros et à les mettre au point. L'exécution du codage est immédiate, sans passer par une phase de compilation. Ceci en fait une base très pratique (et gratuite) pour s'initier à la programmation.

Ce langage est conçu pour écrire, lire et modifier des documents OpenOffice.org en utilisant une API (interface de programmation d'application). Comparativement à d'autres

langages comme Java, qui peuvent aussi piloter OpenOffice.org, il est plus facile de recourir à l'API par le biais d'OOoBasic.

Les macros OOoBasic peuvent être intégrées dans un document, ou intégrées dans OpenOffice.org. Elles sont transportables d'un ordinateur à un autre, et même distribuables sur un réseau de machines. Elles sont exécutables sur tous les systèmes d'exploitation reconnus par OpenOffice.org, un atout important pour les échanges de documents.

Il existe divers moyens pour déclencher une macro : manuellement, en cliquant sur un bouton d'une barre d'outils, en cliquant sur un bouton dans un document, en utilisant une entrée de menu, avec un raccourci clavier, lors de divers événements comme le chargement ou la fermeture du document, ou automatiquement de manière invisible.

OOoBasic et l'API d'OpenOffice.org

OOoBasic n'est pas un langage de programmation autonome : il a besoin d'OpenOffice.org pour s'exécuter. C'est un langage orienté vers l'automatisation de besoins de bureautique, il ne remplace pas les langages généralistes comme C, C++, Delphi, Java et autres. Ceux-ci peuvent piloter OpenOffice.org, mais au prix d'une complexité plus grande et d'une certaine lenteur. En contrepartie, ils disposent de structures de données plus élaborées et généralement de bibliothèques de fonctions utilitaires dans divers domaines. Ils seront donc plutôt employés dans un cadre applicatif plus global que les documents OpenOffice.org.

Cependant, les manipulations de documents OpenOffice.org se font normalement à travers l'API, quel que soit le langage utilisé. Cet ouvrage n'y fait pas exception, et les concepts décrits sont largement réutilisables avec un autre langage.

OOoBasic et VBA

Lorsqu'un utilisateur ou une entreprise décide d'utiliser OpenOffice.org à la place de la suite MS-Office, la conversion des documents Word et Excel existants est effectuée généralement correctement ; en revanche, les macros Visual Basic for Application (VBA™) dans ces documents ne sont pas converties en OOoBasic. La raison est qu'il n'y a pas d'équivalence simple entre les deux langages car les réalisations logicielles sous-jacentes sont différentes bien que les possibilités soient comparables.

Plus précisément, les instructions du langage OOoBasic lui-même sont très similaires et souvent identiques à celles de Visual Basic. Il existe toutefois des différences de détails, qui peuvent nécessiter une modification de l'algorithme. C'est pourquoi vous devriez lire avec attention la deuxième partie de ce livre.

L'équivalent de VBA, c'est-à-dire l'accès aux documents pour les manipuler, se fait par des appels à l'API d'OpenOffice.org depuis OOoBasic. Ici, les objets sont très différents,

et seule une bonne connaissance des deux langages permettra d'écrire un algorithme équivalent. L'effort intellectuel de conversion pouvant être assez important, il est souhaitable de se demander si le document est encore utile, et dans l'affirmative, de déterminer quel est le but à atteindre avec des macros, plutôt que tenter une conversion instruction par instruction ou macro par macro.

MIGRATION De VBA à OOoBasic

James M. Thompson a écrit un document de 60 pages qui présente le portage de Excel/VBA vers Calc/Basic, du point de vue d'un programmeur VBA : « VBA to StarBasic Cross Reference ». Ce document n'existe qu'en anglais, et la dernière version en est disponible sur le site d'OpenOffice.org.

- http://documentation.openoffice.org/HOW_TO/

La version 8 de StarOffice (basée sur la version 2.0 d'OpenOffice.org) offre un outil d'aide à la conversion de macros VBA vers OOoBasic. Nous n'avons pas eu l'occasion de le tester.

Conclusion

OOoBasic est un langage à part entière permettant d'exploiter l'API d'OpenOffice.org. Sa portée s'étend sur toutes les fonctionnalités que peut offrir une suite bureautique. Il permet d'automatiser certaines tâches mais également de créer de nouvelles fonctionnalités en réarrangeant et en utilisant les briques logicielles mises à sa disposition.

L'environnement nécessaire à l'exécution des macros est fourni en standard avec OpenOffice.org, et aucune manipulation supplémentaire ou installation ne sont nécessaires pour les exploiter. Le chapitre suivant va nous permettre de faire nos premiers pas...

2

Enregistrer et exécuter une macro

Dans ce chapitre d'initiation, nous allons passer en revue le moyen le plus simple – mais pas le plus puissant – d'écrire des macros et de les exécuter. De nombreuses notions de base seront introduites et vous pourrez effectuer vos premières manipulations. Toutefois, avant de commencer, soyons prudents...

Les macros et la sécurité

La puissance du langage macro possède son revers : des individus peuvent écrire des documents anodins contenant des macros conçues dans un but malveillant. Les utilisateurs de MS-Outlook, MS-Word et MS-Excel en savent quelque chose.

Depuis la version 1.0, OpenOffice.org offre des moyens de contrôle efficaces ; il les améliore encore avec la version 2.0. Nous allons traiter séparément ces deux versions.

Sécurité des macros sur OOo 1.0 et 1.1

L'utilisateur OpenOffice.org définit les conditions d'exécution des macros à partir du menu Outils>Options>OpenOffice.org>Sécurité, zone Script OpenOffice.org Basic. La figure 2-1 reproduit ce panneau.

Les possibilités de configuration sont nombreuses. Elles sont basées sur le degré de confiance qu'on veut bien accorder :

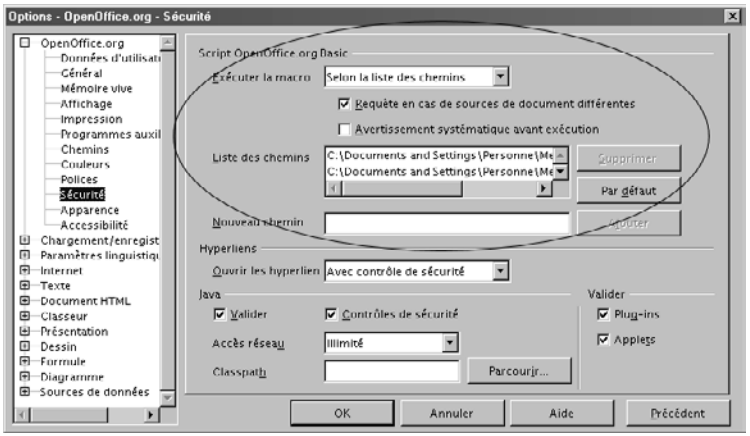


Figure 2–1
Conditions d'exécution des macros

- interdire toute macro ;
- autoriser systématiquement toute macro ;
- ne les autoriser que dans certains dossiers (répertoires) et tous leurs sous-répertoires ;
- afficher un avertissement lors de toute tentative d'exécution d'une macro ;
- afficher l'avertissement seulement si la tentative a lieu dans un répertoire non autorisé.

Le fait d'interdire l'exécution d'une macro n'empêche absolument pas d'ouvrir le document ou de visualiser les instructions de macros avec l'éditeur spécialisé, que nous verrons au chapitre 3.

Sécurité des macros sur OOo 2.0

L'utilisateur OpenOffice.org définit les conditions d'exécution des macros à partir du menu Outils>Options>OpenOffice.org>Sécurité, zone Sécurité des macros. La figure 2-2 reproduit ce panneau.

En fait, tout se passe dans le panneau qui apparaît en cliquant sur le bouton Sécurité des macros. Ce panneau comporte deux onglets ; le premier est reproduit à la figure 2-3.

Les différents niveaux de sécurité

Le niveau de sécurité faible autorise toute macro, quelle que soit l'origine du document. N'utilisez ce niveau que si vous aimez vivre dangereusement.

Le niveau de sécurité moyen vous avertit si le document contenant la macro ne se trouve dans aucun de vos répertoires de confiance (voir l'onglet Sources de confiance). Une bonne sécurité consiste à déclarer quelques répertoires de confiance, ceux où vous savez que certains documents nécessitent des macros. Quand vous récupérez un document

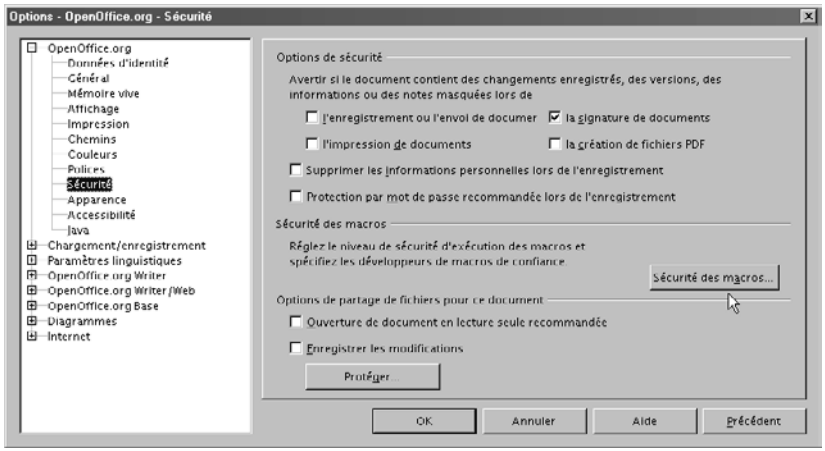


Figure 2–2
Entrée vers le panneau de sécurité des macros

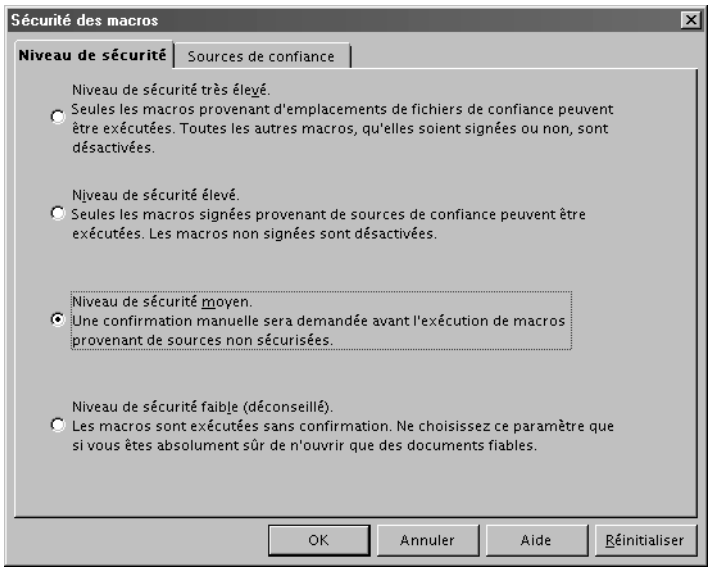


Figure 2–3
Onglet Niveau de sécurité des macros

inconnu, placez-le dans un répertoire ordinaire ; s'il contient des macros, OpenOffice.org vous en avertit (figure 2-4 ou figure 2-5 pour des macros signées) et vous pouvez décider d'autoriser ou non leur exécution.

Là encore, le fait d'interdire l'exécution d'une macro n'empêche absolument pas d'ouvrir le document ou de visualiser les instructions de macros avec l'éditeur spécialisé, que nous verrons au chapitre 3.

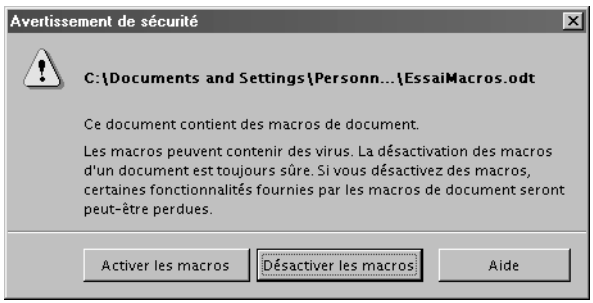


Figure 2–4
Avertissement de macros non signées

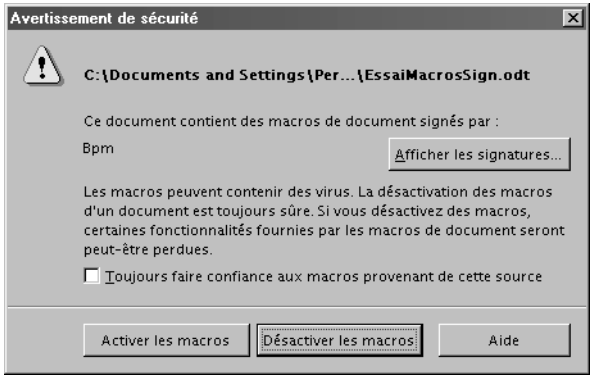


Figure 2–5
Avertissement de macros signées

Le niveau de sécurité **élevé** autorise l'exécution des macros dont le document se trouve dans un des répertoires de confiance. En dehors des répertoires de confiance, les macros sont désactivées et OpenOffice.org vous affiche un message d'avertissement (figure 2-6 ou figure 2-7 pour des macros signées). Dans le cas de la figure 2-7, si vous cochez la case « Toujours faire confiance », tout autre document comportant des macros avec la même signature sera autorisé quelque soit son emplacement.

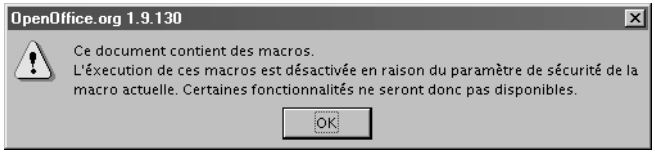
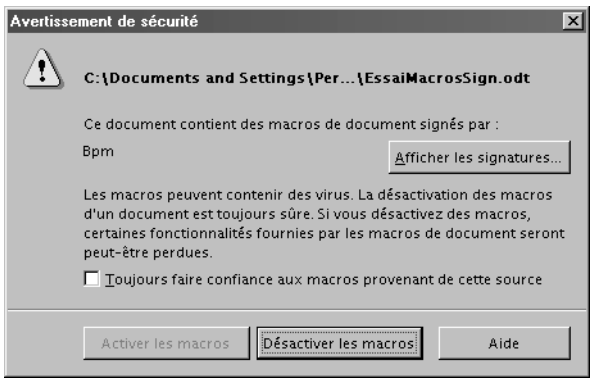


Figure 2–6
Avertissement de macros désactivées

Le niveau de sécurité **très élevé** se base seulement sur les répertoires de confiance. Les macros de documents situés en dehors de ces répertoires sont systématiquement désactivées, qu'elles soient signées ou non, et OpenOffice.org affiche le message d'avertissement de la figure 2-6.

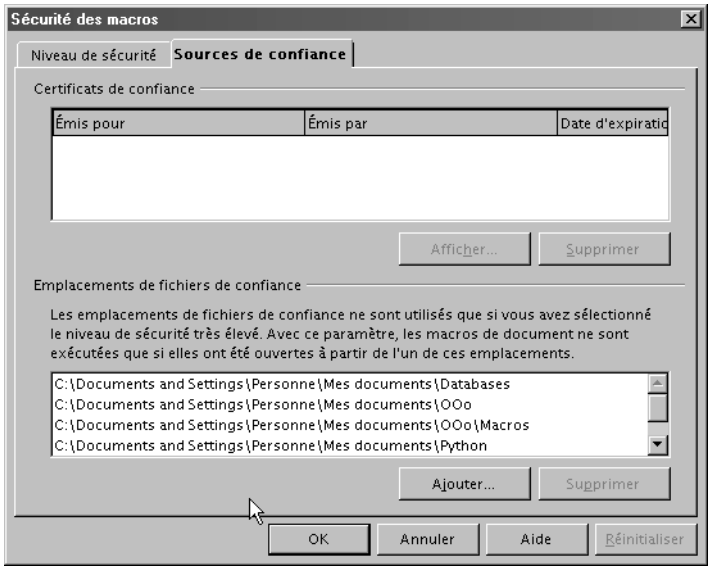
Figure 2–7
Avertissement de macros signées mais désactivées



Les sources de confiance

L'onglet Sources de confiance est reproduit à la figure 2-8.

Figure 2–8
Onglet Sources de confiance



La zone du bas liste les répertoires de confiance. Contrairement à ce qui est indiqué, cette liste est utilisée à partir du niveau de sécurité moyen. Dans la version 2.0.0, la confiance s'applique aux seuls répertoires indiqués, pas à leurs sous-répertoires. À partir de la version 2.0.1, la confiance sera étendue aux sous-répertoires, comme sur les versions 1.0 et 1.1.

La zone du haut liste les certificats de confiance qui représentent des signatures acceptables pour les macros.

Les signatures numériques

La création ou l'importation de certificats permettant de valider une signature numérique nécessite d'autres logiciels comme Firefox et autres navigateurs Internet. Nous ne détaillerons pas ici les procédures.

Un document OpenOffice.org peut être certifié numériquement par un ou plusieurs certificat(s). On certifie un document avec le panneau obtenu par Fichier>Signatures numériques. Lorsque c'est fait, OpenOffice.org calcule une signature numérique du document. Cette dernière est indépendante de la signature éventuelle des macros du document.

De manière similaire, les macros d'un document peuvent être signées avec le panneau de la figure 2-9, obtenu avec le menu Outils>Macro>Signature numérique.

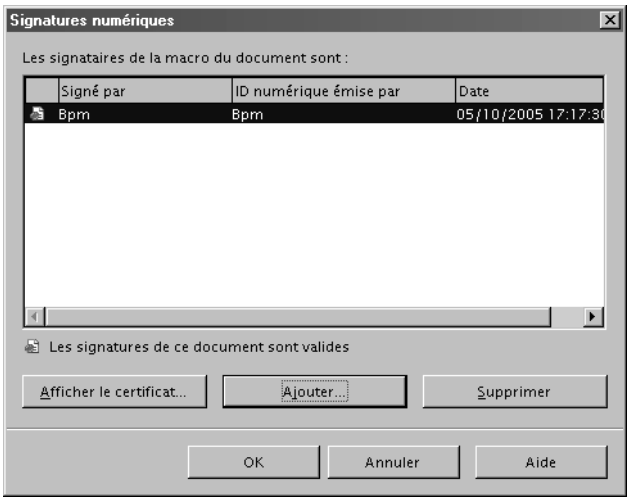


Figure 2–9
Signatures numériques des macros

Le processus est le suivant :

1. sauvez votre document, sans le fermer ;
2. ouvrez le panneau de la figure 2-9 et ajoutez un ou plusieurs certificat(s) ;
3. fermez le document sans le sauver !

Le document peut être copié ou déplacé, il gardera ses signatures.

Plus tard, après toute modification du document, répétez exactement le processus, car la sauvegarde supprime les signatures pour éviter qu'un tiers ne modifie les macros en gardant une apparence de sécurité.

Enregistrer une macro

L'enregistreur de macros est disponible dans OpenOffice.org à partir de la version 1.1. Il enregistre les séquences de manipulations de l'utilisateur sous forme d'une macro, ce qui permet ensuite de reproduire à volonté la même séquence. La méthode est très simple et ne nécessite pas de connaissances de programmation.

Cependant, l'enregistreur de macros souffre de limitations assez sévères :

- Il n'est disponible que sous Writer et Calc.
- Il ne sait que « mimer » des actions de l'utilisateur, et encore, pas toutes.
- Il utilise une fonction non documentée (les « slot ID »).
- Il ne permet pas d'écrire des macros interactives.
- Il produit un codage non optimisé qui est assez difficile à lire.

Par ailleurs, mais c'est le principe d'un tel enregistreur, il n'aide pas à programmer OpenOffice.org et ne peut produire que du codage linéaire (c'est-à-dire qu'il est incapable de faire par exemple une boucle pour répéter une action sur une liste d'objets ou de choisir entre plusieurs alternatives).

La voie qui est développée dans ce livre est plus difficile, mais elle ouvre des perspectives limitées seulement par l'imagination et la compétence de chacun. En apprenant Basic et l'API OpenOffice.org, vous pourrez réaliser des automatisations bien plus sophistiquées qu'avec l'enregistreur de macros.

Il est cependant des cas où l'enregistreur de macros nous sera utile : parfois l'API ne permet pas certaines manipulations que peut réaliser l'enregistreur. Il est alors possible de résoudre la difficulté en combinant un codage Basic avec les instructions créées par l'enregistreur.

Comment enregistrer une macro ?

L'enregistrement est déclenché en cliquant sur Outils>Macros>Enregistrer une macro. À partir de cet instant, toutes les actions sur le document OpenOffice.org contenu dans la même fenêtre sont enregistrées. Vous remarquerez une petite fenêtre en avant-plan : elle vous permet de terminer l'enregistrement.

Ayant cliqué sur cette fenêtre de terminaison de macro, un autre panneau apparaît. Choisissez dans quelle bibliothèque et quel module vous souhaitez sauvegarder votre macro. Nous expliquerons plus loin ces termes et ce panneau. Essentiellement, vous avez le choix entre un module d'une bibliothèque disponible en permanence dans OpenOffice.org et une bibliothèque propre au document en cours. En général, spécialement pour un débutant, il suffit de choisir la bibliothèque `Standard` du document en cours et de cliquer sur le bouton Enregistrer (enregistrer la macro). Un nouveau panneau apparaît, qui vous

demande de choisir le nom du module, par exemple `Module1`. Maintenant, votre macro est écrite dans le module et elle a pour nom `Macro1` ou un nom similaire.

Déclencher facilement une macro

Une fois votre macro enregistrée, vous pouvez l'intégrer dans les mécanismes de votre exemplaire d'OpenOffice.org, ajoutant ainsi une fonctionnalité, soit dans le contexte d'un type de document, soit pour tous les documents.

Comme l'interface utilisateur évolue notablement avec la version 2.0, nous allons consacrer une section particulière à chaque version.

Déclencher une macro avec OOo 1.0 ou 1.1

Exécuter une macro avec le menu Outils

Avec le menu Outils>Macros>Macro, vous obtenez le panneau reproduit à la figure 2-10.

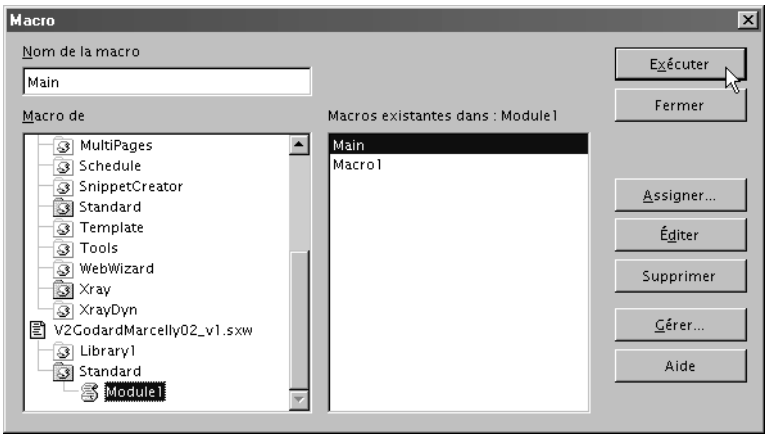


Figure 2–10
Exécuter une macro depuis le panneau Macro

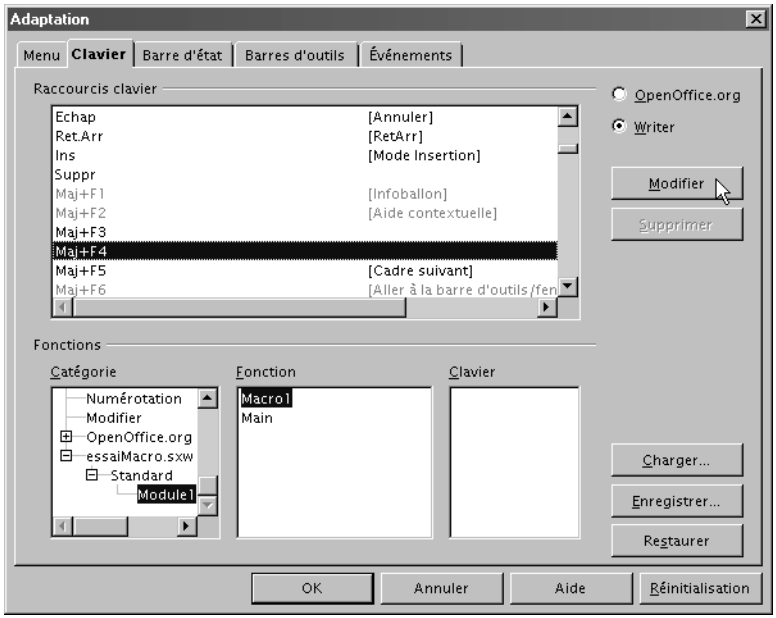
Développez l'arborescence jusqu'à trouver le module contenant votre macro, sélectionnez-la, et cliquez sur le bouton Exécuter.

Exécuter une macro depuis un raccourci clavier

Pour affecter une macro à un raccourci clavier afin de pouvoir ensuite l'exécuter d'une simple combinaison de touches, il faut commencer par choisir un raccourci qui ne soit pas déjà utilisé.

Avec le menu Outils>Adaptation, vous obtenez le panneau Adaptation, qui comporte un onglet Clavier (voir figure 2-11).

Figure 2–11
Ajouter un raccourci clavier



Dans le cadre Raccourcis clavier, choisissez maintenant la combinaison de touches par laquelle vous souhaitez appeler la macro. Pour chaque raccourci, il est indiqué si une commande lui est déjà attribuée.

Une fois que le bon raccourci et la bonne commande sont tous deux sélectionnés dans les panneaux haut et bas, cliquez sur le bouton Modifier. La touche ou combinaison de touches s'affiche dans la fenêtre Clavier et le nom de la macro s'inscrit en face du raccourci.

Cliquez sur OK pour refermer la boîte de dialogue.

Dans votre document, saisissez un paragraphe, puis testez la macro en l'invoquant par son raccourci. À l'appel du raccourci clavier, la macro s'exécute.

Exécuter une macro avec un bouton de barre d'outils

Votre macro peut aussi être lancée en cliquant sur un nouveau bouton dans une des barres d'outils. Utilisez le menu Outils>Adaptation, puis l'onglet Barres d'outils ; choisissez une barre et cliquez sur le bouton Personnaliser.

La figure 2-12 montre l'insertion d'un nouveau bouton dans une barre d'outils, qui déclenchera une macro de la bibliothèque `Standard`.

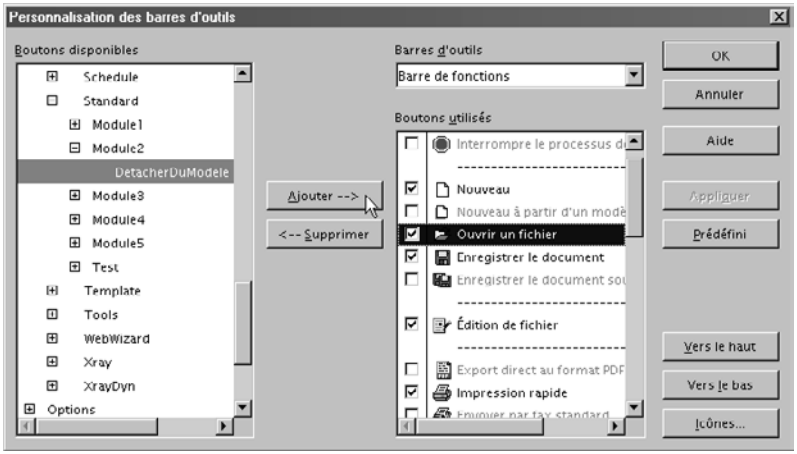


Figure 2–12
Ajouter un bouton dans une barre d'outils

Le même panneau vous permet de choisir ensuite une icône pour votre bouton.

Exécuter une macro par une entrée de menu

OpenOffice.org vous permet de modifier les éléments dans une liste de menu. Il est ainsi possible d'ajouter un élément qui déclenchera une macro. Dans le même panneau **Adaptation**, affichez l'onglet **Menu**, choisissez une position dans la liste des entrées de menus, le module et le nom de votre macro, et cliquez sur **Nouveau**, comme indiqué sur la figure 2-13.

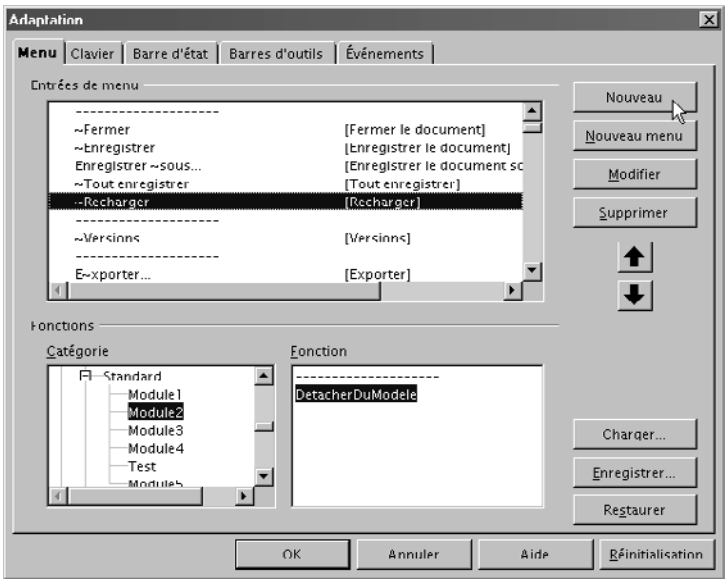


Figure 2–13
Déclencher une macro sur une entrée de menu

Exécuter une macro sur un événement

Pour certains usages, il est parfois intéressant de déclencher automatiquement une macro sur un événement tel que l'ouverture ou la fermeture d'un document. Pour cela, nous utilisons l'onglet Événements du panneau Adaptation (voir figure 2-14).

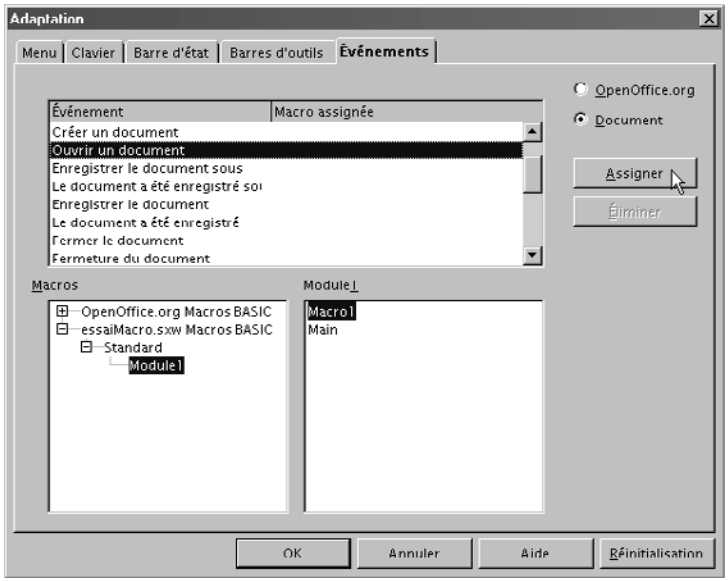


Figure 2–14
Déclencher une macro sur un évènement

Déclencher une macro avec OOo 2.0

Une grande différence entre la version 2.0 et les précédentes est la possibilité d'utiliser différents langages de script : Basic, mais aussi JavaScript, BeanShell, Java compilé, ou Python. Les mécanismes que nous allons décrire permettent d'accéder à des macros écrites dans un quelconque de ces langages.

Exécuter une macro avec le menu Outils

Avec le menu Outils>Macros>Exécuter la macro, vous obtenez le panneau reproduit à la figure 2-15.

Développez l'arborescence jusqu'à trouver le module contenant votre macro, sélectionnez-la, et cliquez sur le bouton Exécuter. Ici, nous avons choisi une macro Basic dans un document.

Ce même panneau permet de choisir une macro écrite dans un autre langage que Basic. Il en existe quelques-unes fournies avec l'installation, dans la branche `Macros OpenOffice.org`. La figure 2-16 vous en montre plusieurs :

- `helloworld.bsh`, macro BeanShell ;

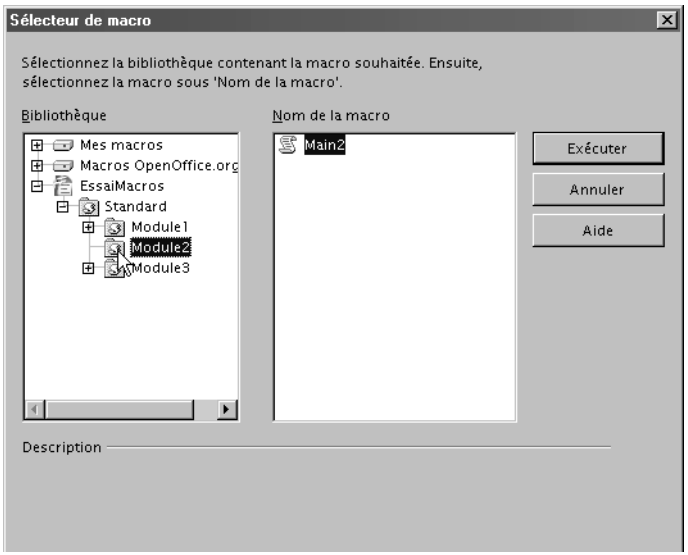


Figure 2–15
Exécuter une macro depuis le panneau Macro

- `helloworld.js`, macro JavaScript ;
- `HelloWorld.printHW`, macro en Java compilé ;
- `HelloWorldPython`, macro en Python.

Vous remarquerez que, sauf pour les macros Basic, il est possible de faire afficher un commentaire descriptif de la macro.

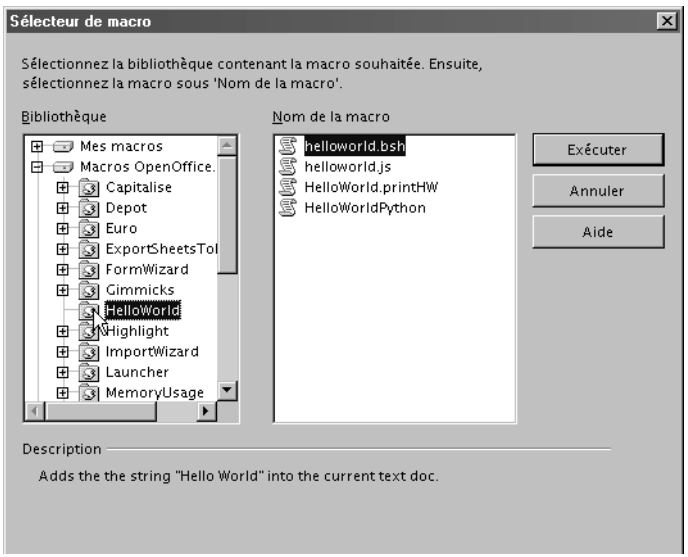
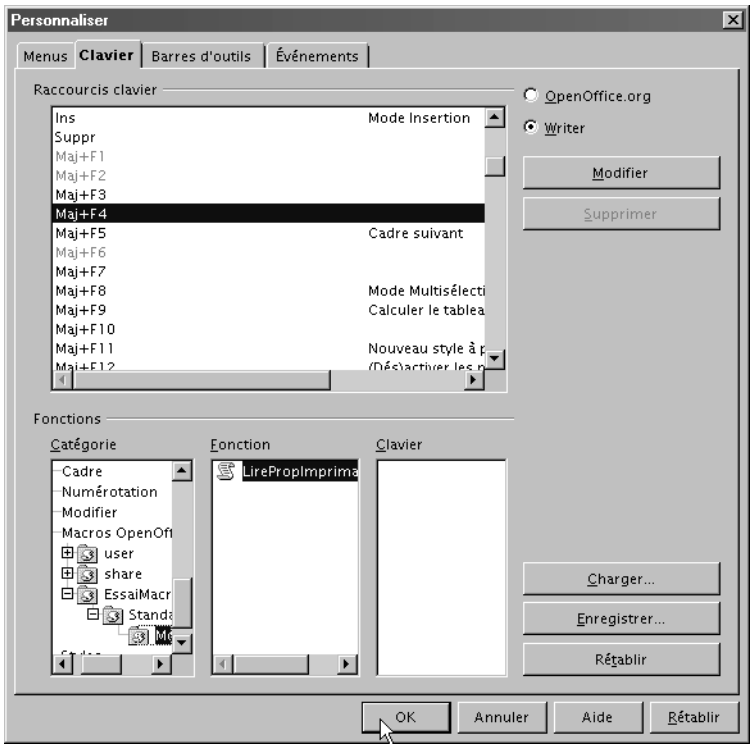


Figure 2–16
Macros écrites en divers langages

Exécuter une macro depuis un raccourci clavier

Avec le menu Outils>Personnaliser, onglet Clavier, vous obtenez le panneau de la figure 2-17.

Figure 2–17
Ajouter un raccourci clavier



Dans le cadre Raccourcis clavier, choisissez la combinaison de touches par laquelle vous souhaitez appeler la macro. Pour chaque raccourci, il est indiqué si une commande lui est déjà attribuée.

Passez maintenant à la minuscule fenêtre Catégorie; utilisez les ascenseurs pour l'explorer. Presque en bas sont regroupées les macros dans la branche Macros OpenOffice.org. Cliquez sur les signes + et vous trouvez :

- `user`, qui regroupe vos macros personnelles disponibles avec tous les documents ;
- `share`, qui regroupe les macros fournies avec OpenOffice.org ;
- `un nom de document`, qui regroupe les macros éventuellement présentes dans le document actuellement ouvert.

En développant l'arborescence, vous finirez par faire apparaître dans la fenêtre Fonctions une liste de noms de macros.

Une fois que le bon raccourci et la bonne commande sont tous deux sélectionnés dans les panneaux haut et bas, cliquez sur le bouton Modifier. La touche ou combinaison de touches s'affiche dans la fenêtre Clavier et le nom de la macro s'inscrit en face du raccourci.

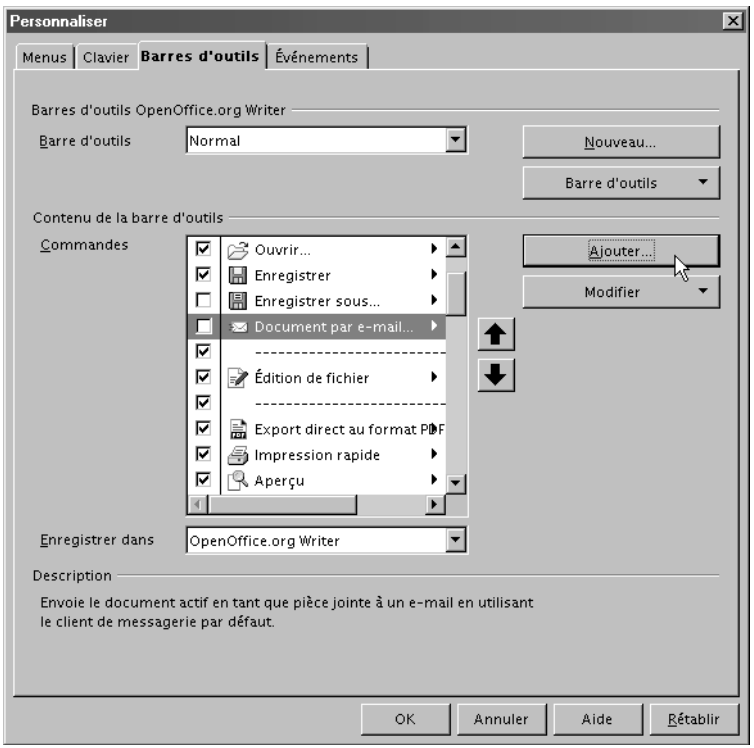
Cliquez sur OK pour refermer la boîte de dialogue.

Dans votre document, saisissez un paragraphe, puis testez la macro en l'invoquant par son raccourci. À l'appel du raccourci clavier, la macro s'exécute.

Exécuter une macro avec un bouton de barre d'outils

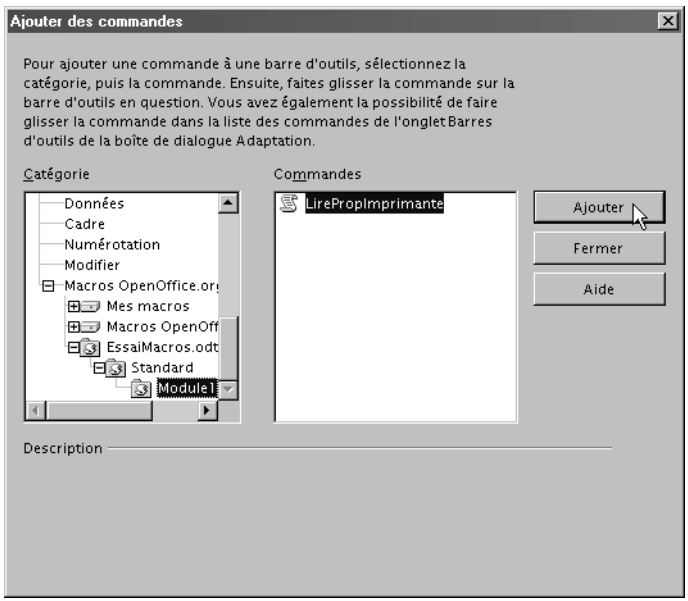
Votre macro peut aussi être lancée en cliquant sur un nouveau bouton dans une des barres d'outils. Dans votre barre d'outils cliquez sur la flèche descendante tout au bout à droite sur la barre. Dans le menu contextuel, choisissez Personnaliser la barre d'outils. Vous obtenez le panneau de la figure 2-18.

Figure 2–18
Personnaliser une barre d'outils



Choisissez la position de votre futur bouton, puis cliquez sur Ajouter. Vous obtenez le panneau de la figure 2-19.

Figure 2–19
Ajouter des commandes



Passez maintenant à la petite fenêtre **Catégorie**; utilisez les ascenseurs pour l'explorer. Presque en bas sont regroupées les macros dans la branche `Macros OpenOffice.org`. Cliquez sur les signes + et vous trouvez :

- `Mes Macros`, qui regroupe vos macros personnelles disponibles avec tous les documents ;
- `Macros OpenOffice.org`, qui regroupe les macros fournies avec OpenOffice.org ;
- `un nom de document`, qui regroupe les macros éventuellement présentes dans le document actuellement ouvert.

En développant l'arborescence, vous finirez par faire apparaître dans la fenêtre **Commandes** une liste de noms de macros. Sélectionnez celle qui vous intéresse et cliquez sur **Ajouter**.

Vous aurez sans doute envie d'affecter une icône à ce nouveau bouton. Dans le panneau de la figure 2-18 cliquez sur le bouton **Modifier** puis choisissez **Changement d'icône** dans le menu.

Exécuter une macro par une entrée de menu

OpenOffice.org vous permet de modifier les éléments dans une liste de menu. Il est ainsi possible d'ajouter un élément qui déclenchera une macro.

Avec le menu Outils>Personnaliser, onglet Menus, vous obtenez le panneau de la figure 2-20.

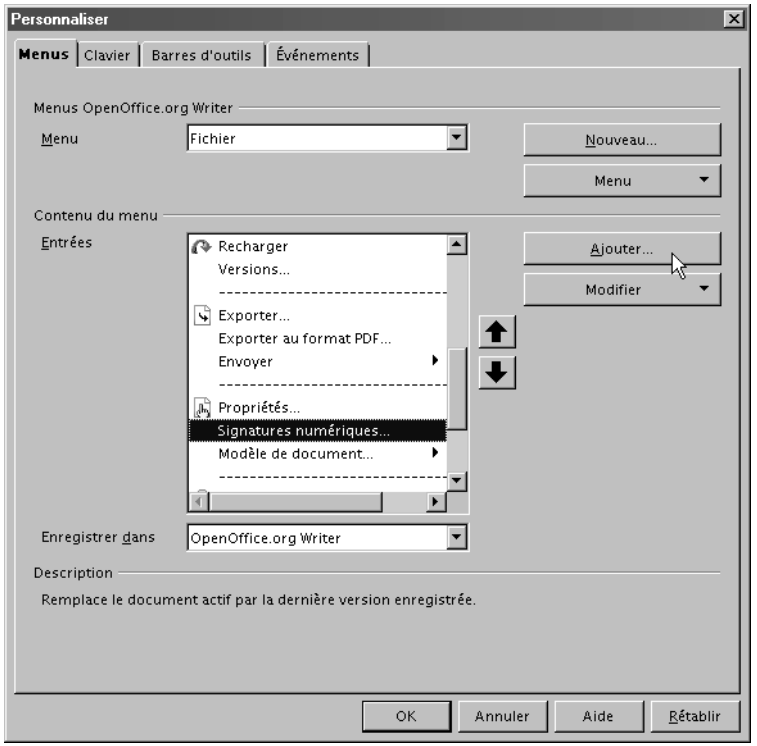


Figure 2–20
Déclencher une macro sur une entrée de menu

Le reste de la procédure est identique au cas précédent : macro sur un bouton de barre d'outils.

Exécuter une macro sur un événement

Pour certains usages, il est parfois intéressant de déclencher automatiquement une macro sur un événement tel que l'ouverture ou la fermeture d'un document. Pour cela, nous utilisons l'onglet Événements du panneau Personnaliser, voir figure 2-21.

Sélectionnez l'événement. Choisissez si l'affectation doit concerner OpenOffice.org ou seulement le document ouvert. Cliquez sur le bouton Assigner la macro. Le panneau Sélecteur de macro apparaît (voir la figure 2-22).

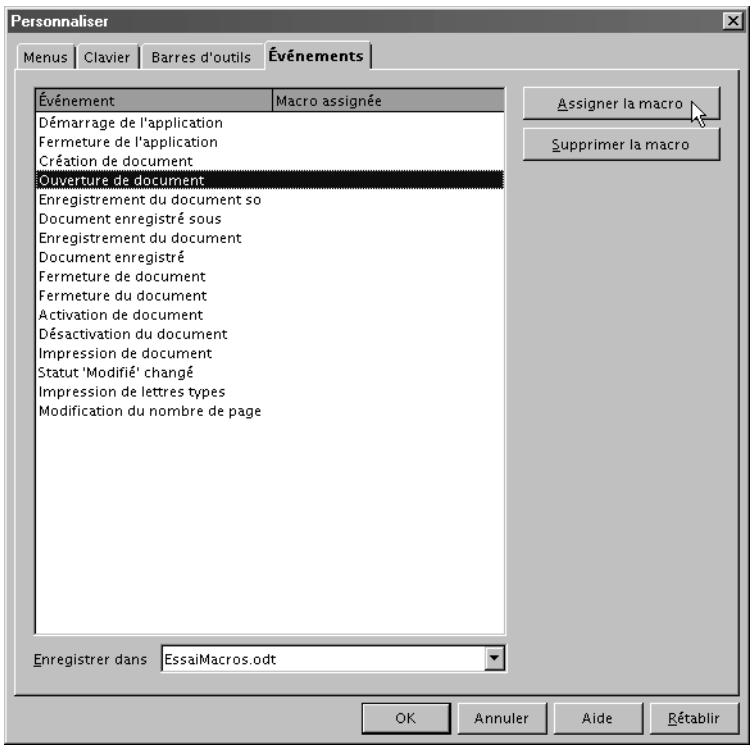


Figure 2–21
Déclencher une macro sur un événement

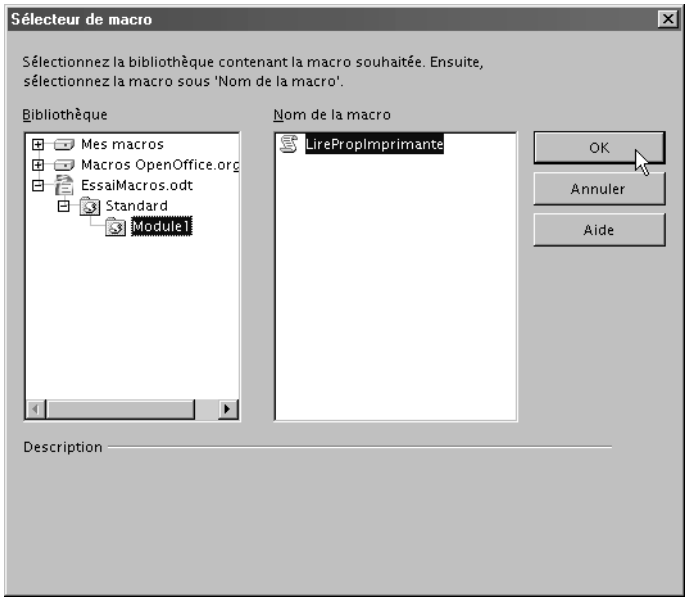


Figure 2–22
Sélectionner une macro pour un événement

Dans la fenêtre Bibliothèque, vous trouvez :

- `Mes macros`, qui regroupe vos macros personnelles disponibles avec tous les documents ;
- `Macros OpenOffice.org`, qui regroupe les macros fournies avec OpenOffice.org ;
- `un nom de document`, qui regroupe les macros éventuellement présentes dans le document actuellement ouvert.

En développant l'arborescence, vous finirez par faire apparaître une liste dans la fenêtre Nom de la macro. Sélectionnez celle qui vous intéresse et cliquez sur OK.

Exécuter une macro en ligne de commande

Il est parfaitement possible de lancer OpenOffice.org pour simplement exécuter une macro, sans passer par l'interface utilisateur. La syntaxe diffère selon le système d'exploitation. Nous décrirons surtout l'appel d'une macro Basic sous MS-Windows, puis nous indiquerons les particularités propres à Linux et dirons un mot des macros d'autres langages.

Cette section nécessite des connaissances de base sur les programmes en ligne de commande (MS-DOS ou Unix Shell)

Sous Windows

Lancer une macro du conteneur soffice

Une macro appartenant à `soffice` est disponible dès qu'OpenOffice.org est chargé. Les éléments suivants doivent être mis successivement dans la ligne de commande :

- le chemin d'accès et le nom de l'exécutable d'OpenOffice.org (`soffice.exe`) (la difficulté est que ce chemin dépend du système d'exploitation,
- de la version et des conditions d'installation d'OpenOffice.org) ;
- le paramètre optionnel `-headless` pour empêcher l'affichage de tout message à destination de l'utilisateur si on souhaite une exécution silencieuse ;
- une URL d'appel de macro, qui se présente sous la forme du mot `macro` suivi des noms de la bibliothèque, du module et de la macro, puis des arguments de cette dernière :

```
macro:///maLib.monModule.maMacro(Arg1, Arg2,...)
```

Dans `soffice`, créez la bibliothèque `maBibli1` et le module `monModule`. Dans ce dernier, écrivez la macro suivante, ou faites un copier-coller dans le fichier `Code03-01.sxw` inclus dans le ZIP téléchargeable :

```
' écrit un fichier texte comportant n fois le caractère c
Sub bavard(c As String, n As Integer)
Dim f As Integer
f = FreeFile

open "C:\essaiMacro.txt" for Output As f
Write #f, Time & " bavard a dit : " & String(n, c)
close #f

msgbox "bavard a terminé"
end sub
```

Cette macro écrit l'heure courante et un texte dépendant de la valeur de ses arguments. Adaptez éventuellement le chemin du fichier résultat `essaiMacro.txt` à votre propre configuration.

Réalisez un fichier *batch* contenant ceci (l'instruction principale doit être écrite en une seule ligne) :

```
rem lancement de la macro
"C:\Program Files\OpenOffice.org1.1.1\program\soffice.exe"
  ➥ "macro:///maBibli1.monModule.bavard(A, 20)"
pause
```

Modifiez éventuellement le chemin d'accès à `soffice` selon votre configuration. Mettez ce fichier dans un répertoire où OpenOffice.org autorise les exécutions de macros.

Arrêtez l'application OpenOffice.org, y compris le lanceur. Exécutez le fichier *batch*. La fenêtre MS-DOS va s'ouvrir, puis le logo de démarrage d'OpenOffice.org va s'afficher et, après un certain temps, le message « bavard a terminé » apparaîtra dans un petit panneau. Cliquez sur OK, puis terminez l'exécution du *batch* en appuyant sur une touche. Vérifiez que le fichier `essaiMacro.txt` est bien écrit.

Réalisez un deuxième fichier *batch*, contenant cette variante :

```
rem lancement de la macro
"C:\Program Files\OpenOffice.org1.1.1\program\soffice.exe"
  ➥ -headless "macro:///maBibli1.monModule.bavard(B, 15)"
pause
```

Arrêtez l'application OpenOffice.org, y compris le lanceur. Exécutez le fichier *batch*. La fenêtre MS-DOS va s'ouvrir et, après un certain temps, affichera l'attente. Aucun autre

message n'est apparu. Terminez l'exécution du *batch* en appuyant sur une touche. Vérifiez que le fichier `essaiMacro.txt` est bien écrit avec cette deuxième commande.

L'argument `-headless` sert donc à indiquer que l'on ne veut pas d'affichage. Les éventuelles boîtes de dialogue de OOo (comme la confirmation de lancement des macros) recevront la réponse par défaut.

Ouvrez un document OpenOffice.org quelconque. Relancez le dernier *batch*. Cette fois-ci, le message « bavard a terminé » s'affiche, malgré l'option `-headless`.

Arguments d'appels de la macro

Comme vous l'avez constaté, le premier argument est considéré comme une chaîne d'un seul caractère ; le deuxième est aussi une chaîne de caractères, mais Basic la convertit en une valeur numérique valide pour le paramètre `n` de la macro.

Sous Windows XP, une chaîne de caractères est transmise telle quelle, sans ajouter de guillemets. Sous d'autres versions de MS-Windows, il peut être nécessaire d'ajouter des guillemets.

Il n'est pas possible de transmettre une chaîne de caractères comportant un guillemet ou une virgule, et les lettres accentuées sont modifiées.

En revanche, il est parfaitement possible de transmettre à la macro les arguments d'appel du fichier *batch*, par exemple :

```
"macro:///maBibli1.monModule.bavard(%1, %2)"
```

Lancer une macro d'un document OpenOffice.org

La méthode diffère sur plusieurs points :

- On doit ajouter le chemin d'accès au document s'il n'est pas déjà chargé.
- L'argument `macro` doit comporter le nom court, sans extension, du document, sous cette forme (attention au nombre de caractères /) :
  `macro://monDoc/maLib.monModule.maMacro(Arg1, Arg2,...)`
- Le document ne se fermera pas automatiquement.

Sur le dernier point, il est possible d'écrire à la fin de la macro une instruction fermant le document, voir le chapitre 10.

Réalisez un fichier *batch* contenant ceci (l'instruction principale doit être écrite en une seule ligne) :

```
rem lancement de la macro
"C:\Program Files\OpenOffice.org1.1.1\program\soffice.exe"
   ➥ "Code03-01.sxw" "macro://Code03-01/Standard.monModule.bavard(W,20)"
pause
```

Modifiez éventuellement le chemin d'accès à `soffice`. Mettez ce *batch* ainsi que le fichier `Code03-01.sxw` (disponible dans le ZIP téléchargeable) dans un même répertoire.

Arrêtez l'application OpenOffice.org, y compris le lanceur. Exécutez le fichier *batch*. La fenêtre MS-DOS va s'ouvrir, puis le document va s'afficher et, après un certain temps, le message « bavard a terminé » apparaîtra dans un petit panneau. Cliquez sur OK. Le document reste ouvert. L'exécution du *batch* ne reprend qu'à la fermeture du document. Vérifiez que le fichier `essaiMacro.txt` est bien écrit.

L'option `-headless` produit le même comportement.

Réalisez un autre fichier *batch*, contenant cette variante :

```
rem lancement de la macro
"C:\Program Files\OpenOffice.org1.1.1\program\soffice.exe"
   ➥ -headless "macro://Code03-01/Standard.monModule.bavard(M,20)"
pause
```

Chargez au préalable le document `Code03-01.sxw`. Exécutez le fichier *batch*. La fenêtre MS-DOS va s'ouvrir et, après un certain temps, le message « bavard a terminé » apparaîtra dans un petit panneau. Cliquez sur OK. Le *batch* reprend son exécution (et pause). Le fichier reste ouvert. Vérifiez que le fichier `essaiMacro.txt` est bien écrit.

Sous Linux

Le principe est le même que sous MS-Windows. Les chemins suivront évidemment la syntaxe Unix et les arguments d'un *shell* seront obtenus par `$1` etc.

Il faut ajouter dans le fichier *shell* une redirection vers un terminal X. Concaténez sur une seule les deux dernières lignes de ce codage. Attention aux guillemets.

```
export DISPLAY=":0.0"
   ➥ /usr/bin/soffice -headless
   ➥ "macro:///maBibli1.monModule.bavard("$1, 15")"
```

Lancer un script autre que Basic

Ceci n'est évidemment possible qu'à partir de la version 2.0 d'OpenOffice.org. Le principe est d'utiliser en argument du programme `soffice.exe` l'URI propre à chaque langage de script. Celle-ci est une chaîne de caractères assez complexe ; nous apporterons quelques précisions à ce sujet au chapitre 19.

Voici un exemple simple : nous appelons le script JavaScript `Hello World` qui se trouve dans `Macros OpenOffice.org`. Il ne nécessite aucun argument (et ne renvoie aucune information).

Sous MS-Windows, le fichier *batch* comportera la ligne suivante (repliée par contrainte d'édition) :

```
"C:\Program Files\OpenOffice.org 1.9.130\program\soffice.exe"
  ➥ "vnd.sun.star.script:HelloWorld.helloworld.js?language=JavaScript&location=share"
```

La deuxième partie de la ligne est l'URI d'appel du script. Vous trouverez d'autres informations sur les scripts au chapitre 19 et dans l'annexe A. Pour que cet exemple fonctionne, on doit avoir ouvert un document texte, car il sera écrit par le script appelé.

Conclusion

Après cette rapide prise en main, nous explorerons au prochain chapitre d'autres manières d'écrire et d'exécuter des macros, notamment depuis l'environnement de développement intégré d'OpenOffice.org qui permet de voir le contenu d'une macro et de l'éditer.

3

Créer et gérer les macros depuis l'EDI

L'environnement de développement intégré (EDI en français ou IDE en anglais) est un outil essentiel pour écrire et mettre au point les macros Basic. Il est recommandé de se familiariser avec ses différents menus et de comprendre leur utilité pour tirer le meilleur parti de Basic.

La version 2 d'OpenOffice.org accepte différents langages de script pour réaliser des macros. Nous décrirons brièvement les mécanismes de l'interface utilisateur qui permettent de les prendre en charge.

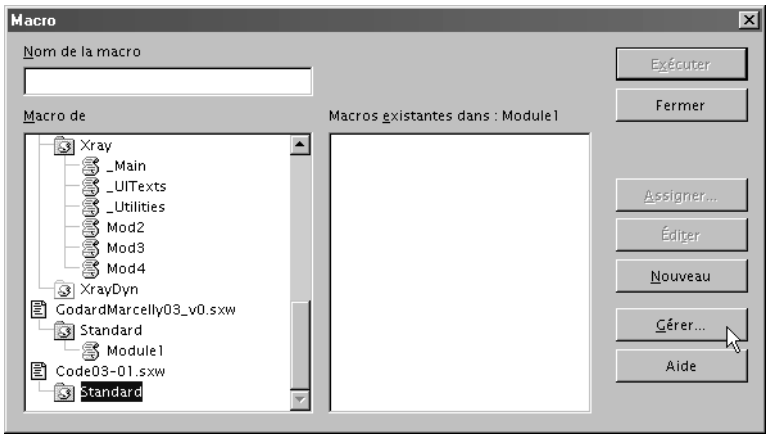
Premiers pas dans l'environnement de développement Basic

Au niveau de l'interface utilisateur, les différences entre la version 2.0 d'OpenOffice.org et les précédentes nécessitent des descriptions séparées, bien qu'au fond il y ait beaucoup de points similaires.

L'environnement de développement sous OOo 1.0 et 1.1

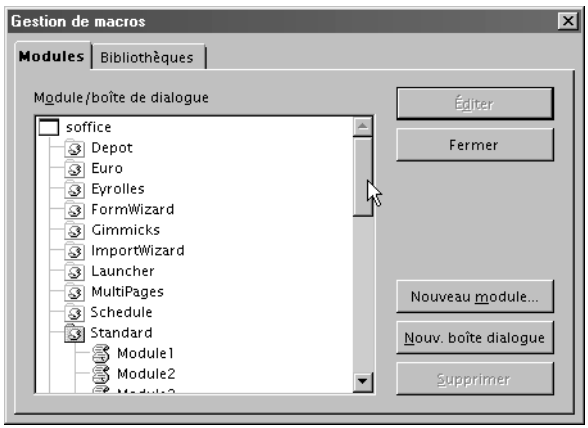
Créez un nouveau document Writer, sauvegardez-le et gardez-le ouvert. Ouvrez le menu Outils>Macros>Macro... Vous obtenez le panneau reproduit à la figure 3-1. L'aspect exact de l'arborescence au-dessous de « Macro de » dépend des macros qui sont disponibles dans votre environnement OpenOffice.org. Vous remarquerez un élément mis en valeur dans la figure, `Standard`, qui est une branche de `Code03-01.sxw`. Ce dernier est le nom du document à partir duquel nous avons déclenché le panneau Macro. `Standard` est le nom de la bibliothèque par défaut ; elle appartient au document et en est une branche.

Figure 3–1
Le premier panneau Macro



Cliquez sur le bouton Gérer.... Vous obtenez un autre panneau Gestion de macros, qui comporte deux onglets, Modules et Bibliothèques (voir la figure 3-2). Restez sur l'onglet Modules.

Figure 3–2
Panneau Gestion de macros

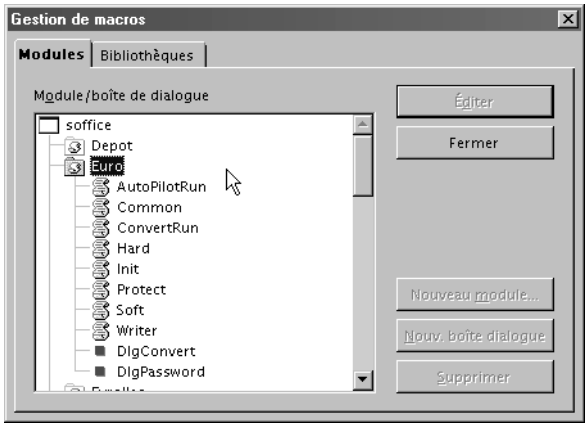


En utilisant l'ascenseur, remontez jusqu'en haut de l'arborescence. Vous verrez un élément racine appelé `soffice`. Il s'agit d'un conteneur de bibliothèques. De la même manière, notre document initial (`Code03-01.sxw` sur la figure précédente) est lui aussi un conteneur de bibliothèques. La particularité de `soffice` est d'être disponible tant que OpenOffice.org est chargé dans votre ordinateur, alors qu'un conteneur propre à un document n'est disponible que si ce document est ouvert. En jouant avec l'ascenseur, vous noterez qu'il existe une bibliothèque nommée `Standard` dans chaque conteneur : cette bibliothèque par défaut est toujours présente, même vide.

Examinez encore l'arborescence de la figure 3-2 : la bibliothèque `Euro` est de couleur blanche pour signifier qu'elle n'est pas chargée actuellement. Pour exécuter une macro d'une bibliothèque, celle-ci doit être chargée.

Double-cliquez sur l'icône de la bibliothèque `Euro`. Une arborescence apparaît, comme sur la figure 3-3, et l'icône devient jaune. Vous venez de charger la bibliothèque `Euro`. Remarquez qu'une fois chargée, il n'est pas possible de la décharger. En fait, il faudrait fermer complètement OpenOffice.org pour cela, ou fermer le document si la bibliothèque appartient à un document.

Figure 3–3
Chargement d'une bibliothèque



L'arborescence de la bibliothèque `Euro` nous montre des noms de modules (on peut effectivement choisir des noms plus parlants que `Module1`, `Module2`, etc.), et des petits carrés verts avec des noms. Ces carrés désignent des boîtes de dialogue, qui permettent d'afficher des informations et de poser des questions à l'utilisateur pour obtenir des réponses de manière semblable aux panneaux d'OpenOffice.org. Nous expliquerons comment créer des boîtes de dialogue au chapitre 15.

Passons maintenant à l'onglet Bibliothèques du panneau Gestion de macros. Il est reproduit sur la figure 3-4 .

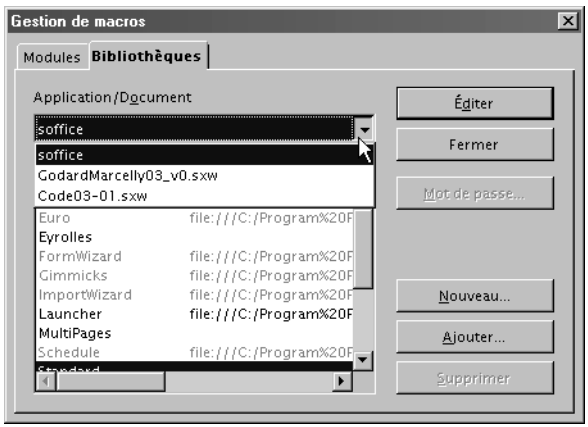


Figure 3–4
Gestion de macros, onglet Bibliothèques

Ce panneau affiche initialement les bibliothèques du conteneur `soffice`. La liste déroulante au-dessus permet de choisir un autre conteneur, en pratique le conteneur de chaque document ouvert. Le bouton Nouveau permet de créer une nouvelle bibliothèque, dont on peut choisir le nom. Le bouton Supprimer efface définitivement la bibliothèque dont la ligne est mise en évidence. Le bouton Mot de passe... permet de chiffrer le contenu des macros d'une bibliothèque, de sorte que seules les personnes connaissant le mot de passe puissent les lire et les modifier. Le bouton Ajouter sert à insérer une bibliothèque existante, comme décrit plus loin dans ce chapitre.

L'environnement de développement sous OOo 2.0

Créez un nouveau document Writer, sauvegardez-le et gardez-le ouvert. Ouvrez le menu Outils>Macros>Gérer les macros>OpenOffice.org Basic. Vous passez par l'arborescence de la figure 3-5, qui vous montre que d'autres langages sont accessibles, et vous obtenez le panneau reproduit à la figure 3-6.

L'aspect exact de l'arborescence au-dessous de « Macro de » dépend des macros qui sont disponibles dans votre environnement OpenOffice.org. Vous aurez toujours deux racines : `Mes macros` et `Macros OpenOffice.org` ; elles contiennent des macros disponibles tant qu'OpenOffice.org est chargé dans votre ordinateur. Il est important de savoir qu'elles correspondent à un seul conteneur de bibliothèques, appelé `soffice` dans les versions précédentes. Ce terme sera employé à divers endroits de ce livre, et il est encore utilisé dans les documents officiels.

Le panneau affiche aussi une racine pour chaque document ouvert : chacune est un conteneur de bibliothèques de macros. Vous remarquerez un élément mis en valeur dans la figure, `Standard`, qui est une branche du document `premierEssai.odt` ; cette biblio-

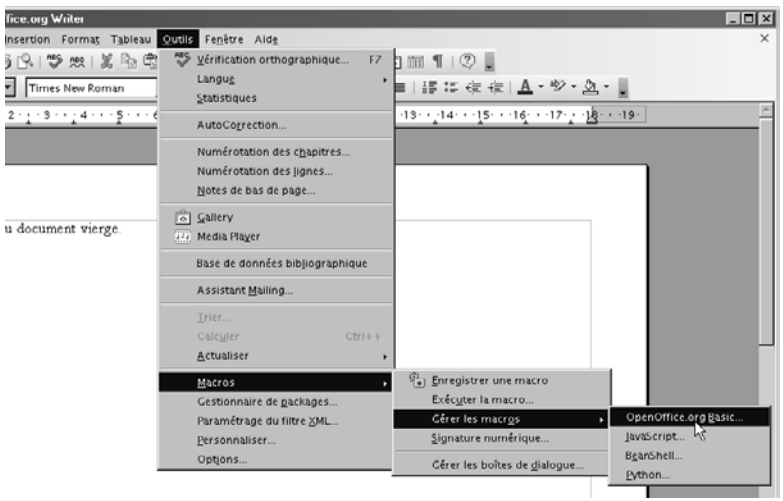


Figure 3–5
L'arborescence de gestion des macros

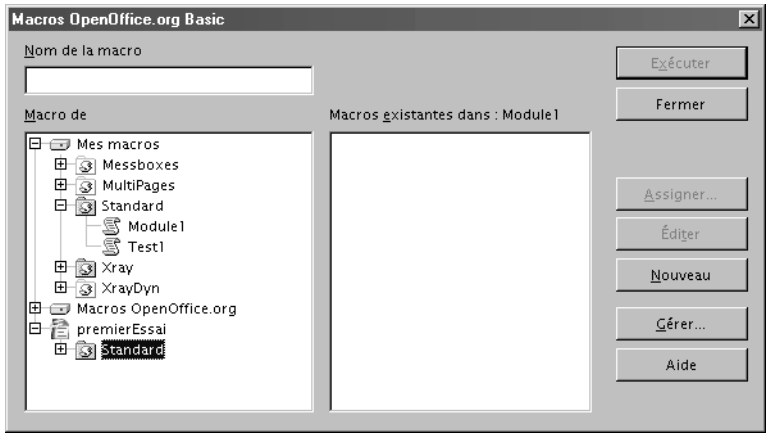


Figure 3–6
Le premier panneau Macros Basic

thèque par défaut est toujours présente, même vide. De même, il existe une bibliothèque nommée `Standard` dans `Mes macros`.

Cliquez sur le bouton **Gérer...**. Vous obtenez un autre panneau **Gestion des macros de OpenOffice.org Basic** qui comporte trois onglets : **Modules**, **Boîtes de dialogue** et **Bibliothèques** (voir la figure 3-7). Restez sur l'onglet **Modules**.

En utilisant l'ascenseur et en cliquant sur les **+**, explorez l'arborescence. Dans notre exemple, nous avons développé le premier niveau de `Macros OpenOffice.org`. Nous avons ainsi affiché la liste de ses bibliothèques.

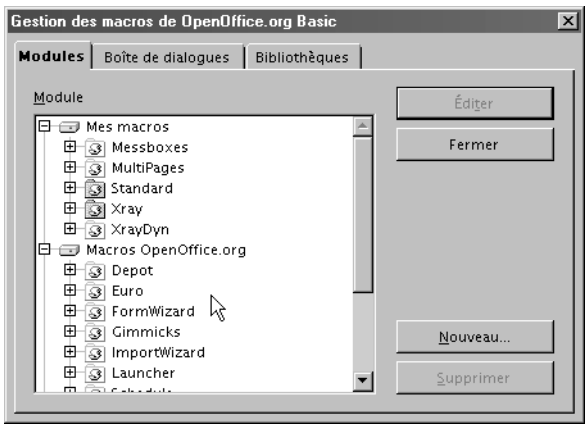


Figure 3–7
Panneau Gestion de macros Basic : modules

Examinez encore l'arborescence de la figure 3-7 : la bibliothèque `Euro` est de couleur blanche pour signifier qu'elle n'est pas chargée actuellement. Pour exécuter une macro d'une bibliothèque, celle-ci doit être chargée.

Double-cliquez sur l'icône de la bibliothèque `Euro`. Une arborescence apparaît, comme sur la figure 3-8 , et l'icône devient jaune. Vous venez de charger la bibliothèque `Euro`. Remarquez qu'une fois chargée, il n'est pas possible de la décharger. En fait, il faudrait fermer complètement OpenOffice.org pour cela, ou fermer le document si la bibliothèque appartient à un document.

L'arborescence de la bibliothèque `Euro` nous montre des noms de modules (on peut effectivement choisir des noms plus parlants que `Module1`, `Module2`, etc.).

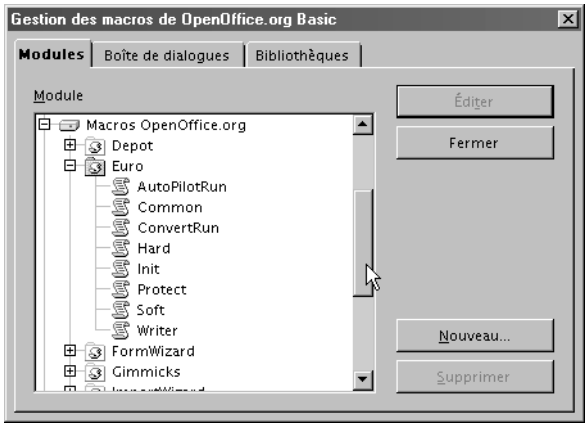


Figure 3–8
Panneau Gestion de macros Basic : modules

Cliquez maintenant sur l'onglet Boîtes de dialogue. Cette page n'affiche que les boîtes existant dans chaque bibliothèque (voir figure 3-9).



Figure 3–9
Panneau Gestion de macros Basic : boîtes de dialogue

Les boîtes de dialogue permettent d'afficher des informations et de poser des questions à l'utilisateur pour obtenir des réponses de manière semblable aux panneaux d'OpenOffice.org. Nous expliquerons comment créer de telles boîtes au chapitre 15. Il est important de savoir que la séparation entre module et boîte de dialogue est purement visuelle : en réalité, ils sont hébergés par une même bibliothèque, comme vous le verrez plus tard en ouvrant l'EDI.

Notez que le contenu des racines `Mes boîtes de dialogue` et `Boîtes de dialogue OpenOffice.org` est disponible tant qu'OpenOffice.org est chargé dans votre ordinateur. Un document peut lui aussi contenir des boîtes de dialogue.

> **REMARQUE** **Boîtes de dialogue universelles**
>
> Avec la version 2.0 d'OpenOffice.org, les boîtes de dialogue ne sont pas réservées aux seules macros Basic : elles sont un mécanisme utilisable depuis tout langage de script.

Passons maintenant à l'onglet Bibliothèques du panneau Gestion de macros. Il est reproduit sur la figure 3-10.

Ce panneau affiche initialement les bibliothèques de Mes macros et boîtes de dialogue. La liste déroulante au-dessus permet de choisir les bibliothèques communes Macros et boîtes de dialogue OpenOffice.org ou celles de chaque document ouvert. Le bouton Nouveau permet de créer une nouvelle bibliothèque, dont on peut choisir le nom. Le bouton Supprimer élimine définitivement la bibliothèque dont la ligne est mise en évidence. Le bouton Mot de passe... permet de chiffrer le contenu des macros d'une bibliothèque, de

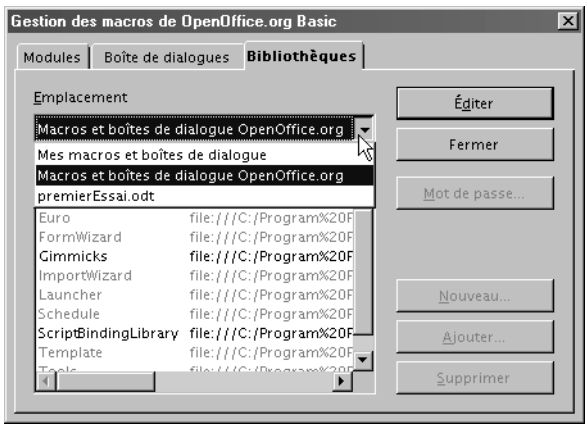


Figure 3–10
Gestion de macros, onglet Bibliothèques

sorte que seules les personnes connaissant le mot de passe puissent les lire et les modifier. Le bouton **Ajouter** sert à insérer une bibliothèque existante, comme décrit plus loin dans ce chapitre.

Informations complémentaires

Résumons-nous :

- Il existe plusieurs conteneurs de bibliothèques : `soffice` et un conteneur par document.
- Dans la version 2.0 d'OpenOffice.org, le conteneur `soffice` est visuellement séparé en `Mes macros` et `Macros OpenOffice.org`. Les boîtes de dialogue de `soffice` sont visuellement séparées en `Mes boîtes de dialogue` et `Boîtes de dialogue OpenOffice.org`.
- Chaque conteneur peut renfermer plusieurs bibliothèques, dont au moins une bibliothèque appelée `Standard`.
- Chaque bibliothèque peut renfermer plusieurs modules (jusqu'à 16 000 selon l'aide en ligne ; nous avons seulement testé avec 250 modules de plus de 30 000 caractères chacun !).
- Chaque module peut renfermer plusieurs macros et comporter jusqu'à 65 000 caractères.

Pourquoi une telle organisation à tiroirs ? Pour des raisons de souplesse et de modularité. Un utilisateur chevronné peut avoir besoin de nombreuses macros, pour différents cas d'utilisation. Des macros concernant un domaine commun seront regroupées dans différents modules d'une même bibliothèque. D'une bibliothèque à l'autre, il est possible de

reprendre les mêmes noms de macros sans risquer d'interférence ; donc, chaque bibliothèque peut être développée de manière totalement indépendante.

Pourquoi un conteneur `soffice` ? Si vous avez besoin d'une macro disponible dans tous les documents, par exemple une macro qui permet d'intervertir deux lettres sur un texte, vous la mettrez dans une bibliothèque de `soffice`. Si votre macro est utile dans un seul document, vous la mettrez dans ce dernier.

Le conteneur `soffice` est composé de deux sous-ensembles, mémorisés dans des fichiers informatiques de deux répertoires différents : `user/` et `share/`. La version 2.0 distingue visuellement ces deux sous-ensembles en les nommant `Mes macros` et `Macros OpenOffice.org`. Le premier sous-ensemble correspond à vos propres macros, que vous souhaitez rendre disponibles au niveau de votre application OpenOffice.org. Le second correspond à des macros stockées au niveau administrateur (installation reseau -net pour OOo 1.x et par défaut pour OOo 2.0).

> **À RETENIR Bibliothèque Standard**
>
> Chaque bibliothèque `Standard` est systématiquement chargée par OpenOffice.org, en même temps que le document correspondant. La bibliothèque `Standard` du conteneur `soffice` est toujours chargée. Ceci rend disponibles toutes les macros de ces bibliothèques. OpenOffice.org ne charge pas systématiquement toutes les bibliothèques pour éviter de consommer inutilement la mémoire du système.

Gérer les bibliothèques de macros Basic

Nous allons vous montrer maintenant les possibilités qu'offre le panneau `Gestion de macros`. Nous ne distinguerons pas toujours le cas de la nouvelle version 2.0 car tout est très similaire.

Renommer une bibliothèque

Il est possible de renommer une bibliothèque existante (autre que `Standard`) à partir de l'onglet `Bibliothèques`. D'abord, assurez-vous que la bibliothèque est chargée, comme indiqué plus haut. Ensuite, cliquez sur le nom de la bibliothèque à renommer. Toute la ligne est mise en évidence. Cliquez à nouveau après une seconde, seul le nom de la bibliothèque est alors mis en évidence : vous pouvez maintenant le modifier. Cliquez ensuite ailleurs pour valider le nom.

La méthode indiquée pour renommer une bibliothèque, la seule disponible, ne doit se faire que sur une bibliothèque chargée ! Sinon, vous risquez de perdre tous les modules de

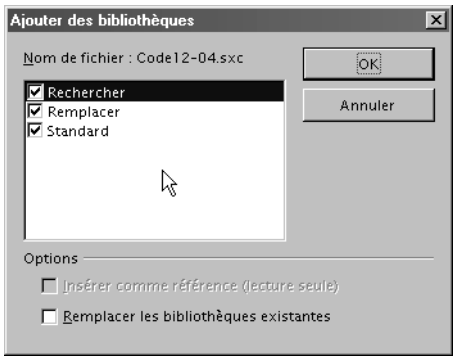
la bibliothèque et les macros qu'ils contiennent. Un rapport d'anomalie a été émis à ce sujet. Prenez vos précautions, et ayez une copie de sauvegarde de vos chers documents.

Ajouter une bibliothèque

Affichez l'onglet Bibliothèques du panneau Gestion de macros. Prenez soin de choisir le conteneur souhaité dans la liste déroulante : un document, ou `soffice` ou `Mes Macros` (version 2.0).

Cliquez sur le bouton Ajouter... ; un panneau de recherche de fichier s'affiche. Recherchez sur votre ordinateur un document OpenOffice.org contenant une bibliothèque de macros. Dans l'exemple de la figure 3-11, le document contient trois bibliothèques : `Rechercher`, `Remplacer` et `Standard`.

Figure 3–11
Ajouter une bibliothèque



La case à cocher devant chaque bibliothèque vous permet de la charger ou non dans le conteneur courant. La case à cocher Remplacer les bibliothèques existantes écrase le contenu entier d'une bibliothèque de même nom dans le conteneur cible ; sinon, la bibliothèque existante restera inchangée. Ceci est particulièrement important pour la bibliothèque Standard, qui contient souvent diverses macros. C'est pourquoi il est préférable de grouper les macros dans des bibliothèques spécialisées.

Il existe une deuxième possibilité de chargement, utile dans certains cas où la bibliothèque source est seulement accessible sous forme de répertoire sur un disque. C'est en particulier le cas des bibliothèques de `soffice` (ou équivalent dans la version 2.0). On cherchera d'abord un répertoire du nom de la bibliothèque source ; dans ce répertoire, on choisira le fichier nommé `script.xlb` ou `dialog.xlb`.

Avec les versions 1.0 et 1.1 d'OpenOffice.org, les bibliothèques de `soffice` se trouvent dans l'arbre des fichiers d'installation d'OpenOffice.org, soit dans `user/basic`, soit dans `share/basic`.

Avec la version 2.0, les bibliothèques de `Mes Macros` se trouvent dans la branche `user/basic/` et celles de `OpenOffice.org Macros` se trouvent dans la branche `share/basic/`.

Avec la version 2.0, la branche `user/` se trouve dans le répertoire d'application de l'utilisateur. Sous MS-Windows XP, il s'agit de :

```
C:\Documents and Settings\monNom\Application Data\OpenOffice.org2\user
```

Copier une bibliothèque

Il est possible de copier une bibliothèque de `soffice` vers un document, et inversement. Ce procédé est utile comme sauvegarde et comme moyen de transfert de PC à PC.

La description précédente vous indique la marche à suivre dans les deux cas. Pour copier une bibliothèque d'un document vers un autre document il suffit de combiner les deux descriptions.

Protéger une bibliothèque par mot de passe

Par défaut, vos bibliothèques et macros sont consultables et éditables par l'utilisateur. Si dans la majeure partie des cas, cela ne pose pas de problème particulier, il est parfois nécessaire de limiter cet accès par un mot de passe pour protéger des données confidentielles qui s'y trouveraient (un mot de passe d'un document par exemple) ou pour éviter des altérations par mégarde.

Pour protéger une bibliothèque, il suffit de se rendre dans le gestionnaire de macros, onglet Bibliothèque. Si une bibliothèque est choisie, le bouton Mot de passe... de la figure 3-4 est accessible. En cliquant dessus, vous définissez un nouveau mot de passe ou vous le modifez. Une clé apparaît à coté du nom de la bibliothèque dans le gestionnaire de macros.

Le contenu des fichiers sera alors chiffré (crypté). Pour que la sécurité soit effective, il faut fermer le document contenant cette bibliothèque, ou fermer OpenOffice.org si la bibliothèque est dans `soffice`. Les macros contenues seront ainsi exécutables, mais un mot de passe sera demandé pour accéder à l'EDI et en voir le code source.

Depuis un bouton, un menu, un raccourci, on peut appeler une macro située dans une bibliothèque protégée. Il n'est pas nécessaire de connaître le mot de passe. La bibliothèque sera chargée automatiquement, mais ne sera pas pour autant visible dans l'EDI.

De même, on peut appeler depuis une macro une macro se trouvant dans une bibliothèque protégée, sans utiliser le mot de passe. Il suffit de charger la bibliothèque, et de connaître le nom de la macro et ses arguments.

BOGUE Protection d'une bibliothèque de document

En version 1.1, le système de protection de bibliothèque de document fonctionne mal (perte du mot de passe). Il est corrigé dans la version 2.0. La protection d'une bibliothèque de `soffice` est correcte dans les différentes versions.

Déplacer un module d'une bibliothèque à une autre

Affichez l'onglet Modules du panneau Gestion de macros. Par un simple glisser-déplacer à la souris, vous pouvez déplacer un module d'une bibliothèque vers une autre, ainsi que le montre la figure 3-12. Dans cet exemple, les deux bibliothèques sont dans deux documents différents.

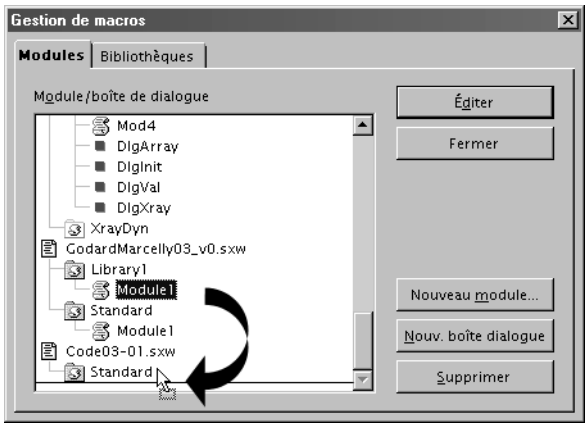


Figure 3–12
Transfert d'un module entre bibliothèques

Bien entendu, la bibliothèque cible ne doit pas déjà contenir un module de même nom. En appuyant sur la touche Ctrl, la même manipulation effectue une copie du module (l'icône de transfert affiche un signe +).

Les boîtes de dialogue peuvent de même être transférées ou dupliquées.

Créer, supprimer un module ou une boîte de dialogue

Ces opérations sont aussi réalisables dans la fenêtre de l'éditeur de macros, que nous décrivons plus loin.

Avec OOo 1.0 ou 1.1

Affichez l'onglet Modules du panneau Gestion de macros.

Pour créer un nouveau module, cliquez sur la bibliothèque qui vous intéresse, puis sur le bouton Nouveau module.... Choisissez le nom du module selon vos souhaits. Pour créer une nouvelle boîte de dialogue, cliquez sur le bouton Nouv. boîte dialogue (les traducteurs ont eu du mal à caser le titre sur le bouton).

Pour supprimer un module ou une boîte de dialogue, cliquez sur cet élément pour le mettre en évidence, puis cliquez sur le bouton Supprimer.

Avec OOo 2.0

Pour créer un nouveau module, affichez l'onglet Modules du panneau Gestion des macros, cliquez sur la bibliothèque qui vous intéresse, puis sur le bouton Nouveau module.... Choisissez le nom du module selon vos souhaits.

Pour créer une nouvelle boîte de dialogue, affichez l'onglet Boîtes de dialogue du panneau Gestion des macros, puis cliquez sur le bouton Nouveau.

Pour supprimer un module ou une boîte de dialogue, cliquez sur cet élément pour le mettre en évidence, puis sur le bouton Supprimer.

Gérer les macros Basic

Revenez sur le premier panneau Macro de la figure 3-1 (figure 3-6 en version 2.0). Les actions effectuées à partir de ce panneau concernent seulement les macros Basic.

Si la bibliothèque mise en évidence dans la partie gauche ne possède pas de module, le bouton Nouveau en crée un dans cette bibliothèque. Si le module mis en évidence dans la partie gauche ne possède pas de macro, le bouton Nouveau en crée une dans le module et lui donne pour nom `Main` (terme anglais pour *Principal*). De plus, la fenêtre de l'éditeur de macros s'affiche, montrant cette macro :

```
REM  *****  BASIC  *****

Sub Main

End Sub
```

Il s'agit simplement d'une macro vide. Vous pouvez changer le nom de la macro et écrire ce que vous souhaitez.

Fermez cette fenêtre et réaffichez le panneau Macro. La partie droite de celui-ci liste les macros qui se trouvent dans le module sélectionné à gauche. Cliquez sur l'une d'elles pour la sélectionner (figure 3-13). En passant, constatez que le bouton Nouveau a disparu, remplacé par un bouton Supprimer.

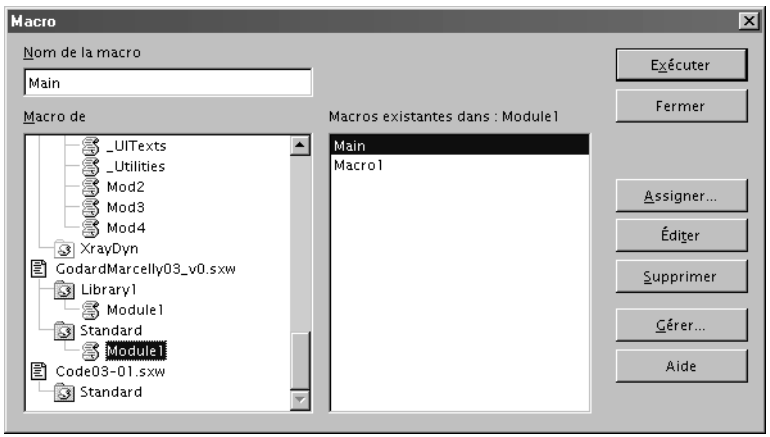


Figure 3–13
Panneau Macro avec une macro sélectionnée

Le bouton Exécuter permet de lancer la macro sélectionnée. Le bouton Assigner... ouvre le panneau du menu Outils>Adaptation (Outils>Personnaliser en version 2.0) afin de faire déclencher la macro sur un événement, sur une entrée de menu, sur un raccourci clavier, ou en déclenchant un bouton de barre d'outils.

Le bouton Supprimer élimine la macro sélectionnée.

Le bouton Éditer ouvre la fenêtre de l'éditeur de macros Basic, que nous allons découvrir maintenant.

La fenêtre d'édition de macros Basic

La figure 3-14 présente la fenêtre de l'éditeur de macros, appelé aussi Environnement de Développement Intégré, ou EDI. La fenêtre est similaire en version 2.0 mis à part l'aspect et l'ordre des boutons de barres d'outils. Cette fenêtre affiche le contenu du module `Module1` de la bibliothèque `Standard` du document `Code03-01.sxw`. Ces informations apparaissent dans la ligne d'état en bas de la fenêtre.

Nous voyons aussi un onglet intitulé Module1 et des boutons pour se déplacer d'un onglet à l'autre. Chaque onglet correspond à un module de la bibliothèque en cours. Ici, nous n'en avons qu'un seul.

Le cœur de la fenêtre est une zone d'édition comportant toutes les instructions et les commentaires du module. Ici, nous avons deux macros créées automatiquement, `Main` et `Macro1`, précédées d'une ligne commentaire qui rappelle que ce qui suit est du langage Basic. Un éditeur de texte est similaire à un traitement de texte, excepté qu'on ne peut employer aucune mise en forme (gras, italique, souligné, couleurs).

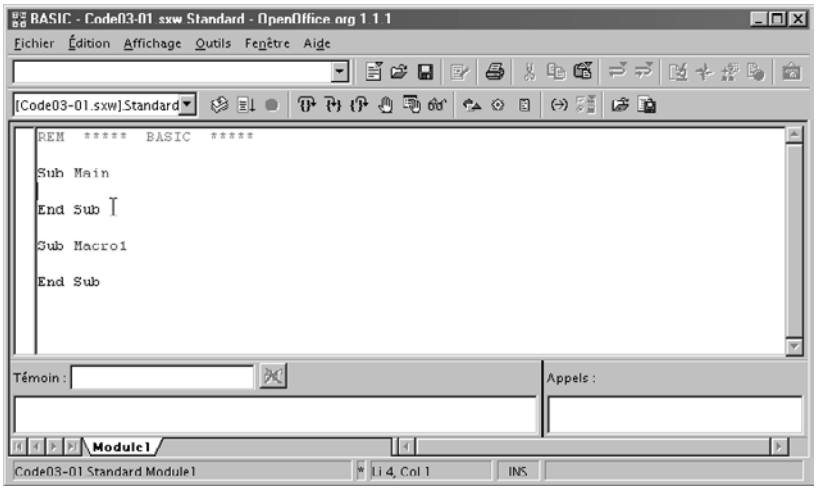


Figure 3–14
La fenêtre de l'éditeur de macros

La coloration syntaxique

Ce n'est pas visible dans le livre, mais certains mots de la zone d'édition sont colorés : l'éditeur Basic offre la coloration syntaxique, c'est-à-dire qu'il reconnaît instantanément des mots-clés du Basic et leur affecte une couleur particulière, sans action particulière de votre part. C'est très pratique à l'usage car cela permet une première vérification du codage. Par exemple, tout commentaire, comme la première ligne, est reconnaissable. Un commentaire est ignoré à l'exécution, donc un codage de cette couleur ne sera jamais exécuté (c'est parfois volontaire).

Notez que, parfois, Basic colore à tort certains mots, car il les reconnaît comme des mots réservés, alors que dans le contexte leur signification est autre.

Si les couleurs proposées vous rendent la lecture difficile, vous pouvez les changer à partir du menu Outils>Options>OpenOffice.org>Apparence. Dans la zone Couleurs personnalisées, utilisez l'ascenseur vers le bas pour voir la section Mise en évidence de la syntaxe de base. Il s'agit d'une mauvaise traduction, corrigée sur la version 2.0 où on lit : Mise en évidence de la syntaxe du Basic. Pour chaque élément syntaxique, vous pouvez choisir la couleur à utiliser.

Ma première macro Basic

Les deux macros de la figure précédente sont parfaitement constituées et peuvent être exécutées. Seulement, elles ne font rien de particulier. Nous allons faire quelques modifications avec l'éditeur de macros.

Remplacez le mot `Main` par un nom de macro plus significatif : `DireBonjour`.

Sur la ligne vide suivante, tapez deux espaces et le texte suivant :

```
  info = "Bonjour !"
```

Utilisez bien le caractère guillemet (au-dessous de la touche 3 du clavier principal). Au fur et à mesure que vous tapez, le texte apparaît d'une certaine couleur à partir du premier guillemet, puis toute la séquence change de couleur quand vous tapez le guillemet final. Ceci est encore un avantage de la coloration syntaxique : elle vous permet de vérifier que votre chaîne de caractères est correctement écrite. Nous venons d'affecter la chaîne de caractères « Bonjour ! » à la variable info.

Placez le curseur en bout de ligne et tapez sur la touche Entrée. Vous constatez que votre curseur se retrouve sur une nouvelle ligne, mais après deux espaces automatiquement insérés. C'est encore une facilité de l'éditeur de macro : l'indentation automatique en fonction de la ligne précédente. Elle produit une sorte de structuration des instructions successives, qui facilite la relecture. En Basic, les espaces et tabulations peuvent être employés entre des mots sans influence sur l'exécution. Naturellement, vous pouvez effacer les espaces si nécessaire. Saisissez donc cette ligne :

```
  MsgBox info
```

Elle a pour fonction d'afficher un petit panneau contenant notre salutation et un bouton OK pour le refermer. L'instruction `MsgBox` est assez sophistiquée, mais il est inutile d'apprendre par cœur les différents arguments. Faites un simple clic sur le mot `MsgBox` et tapez sur la touche F1 : l'aide en ligne apparaît, avec son index positionné sur ce mot. Choisissez l'entrée instruction et appuyez sur la touche Entrée pour visualiser la syntaxe de l'instruction.

Voilà, nous avons créé une macro qui vous dit bonjour. Il n'y a plus qu'à l'exécuter, ce qui peut se faire de différentes façons.

Exécuter une macro depuis l'éditeur Basic

Nous avons déjà vu une manière d'exécuter une macro, avec le panneau Macro. Cependant, il est possible de l'exécuter depuis l'éditeur ; cela nous ouvrira davantage de possibilités.

Cette macro étant la première du module, nous pouvons l'exécuter en cliquant sur le bouton Exécuter (deuxième bouton dans la deuxième barre d'outils). Le résultat est reproduit à la figure 3-15.

Figure 3–15
Message affiché par notre macro



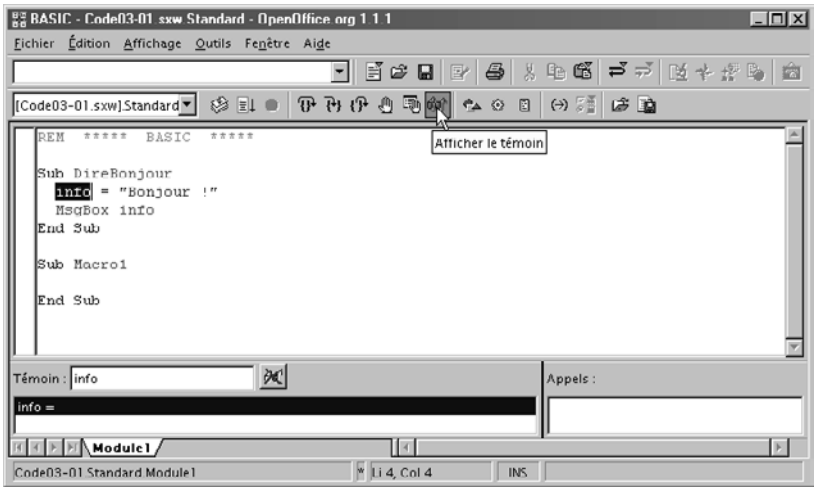
Si vous avez bien observé avant de cliquer sur le bouton OK, vous avez remarqué qu'un « voyant rouge » s'est allumé sur la barre d'outils. Il vous indique qu'une macro est en cours d'exécution. Dans les cas de bouclage infini (si vous avez fait une erreur dans votre programme), vous pourrez arrêter l'exécution en cliquant sur ce voyant. Dans certains cas malheureusement, ce ne sera pas possible...

Aides à la mise au point

Voir le contenu d'une variable

Placez votre souris au-dessus du mot « info » et cliquez pour positionner le curseur dessus. Ensuite, cliquez sur l'icône représentant une paire de lunettes, comme indiqué à la figure 3-16.

Figure 3–16
Afficher un témoin



Vous venez de demander à l'éditeur de vous afficher en permanence la valeur de la variable `info`. On appelle ceci un témoin. Pour le moment, cette variable n'a pas de valeur car la macro ne s'exécute pas. Il est nécessaire d'arrêter temporairement la macro pendant son exécution.

La première méthode pour cela consiste à avancer ligne par ligne dans l'exécution du codage. Il suffit de cliquer sur l'icône **Étape par étape**. En cherchant cette icône, vous aurez remarqué deux autres icônes similaires dont vous comprendrez l'utilité à l'usage. Cliquez donc sur l'icône **Étape par étape**, une fois, encore une fois, etc. Une flèche jaune en marge vous indique la prochaine instruction à exécuter. Le voyant rouge reste allumé, car la macro est en cours d'exécution.

Observons la zone des témoins. Notre variable `info` commence par afficher `<Out of Scope>`, ce qui veut dire qu'elle n'est pas significative. Puis, sa valeur s'affiche quand la flèche jaune a dépassé l'instruction d'affectation. Cliquez encore sur l'icône. Le message s'affiche. Cliquez sur le bouton **OK**. La flèche jaune pointe sur la dernière ligne de la macro, l'exécution n'est pas encore terminée. Cliquez une dernière fois sur l'icône **Étape par étape**.

La version 2.0 d'OpenOffice.org permet de visualiser le contenu d'une variable objet de l'API. Ces objets peuvent être extrêmement sophistiqués, comme nous le verrons à partir du chapitre 10. La figure 3-17 présente un exemple de l'affichage d'un tel objet.

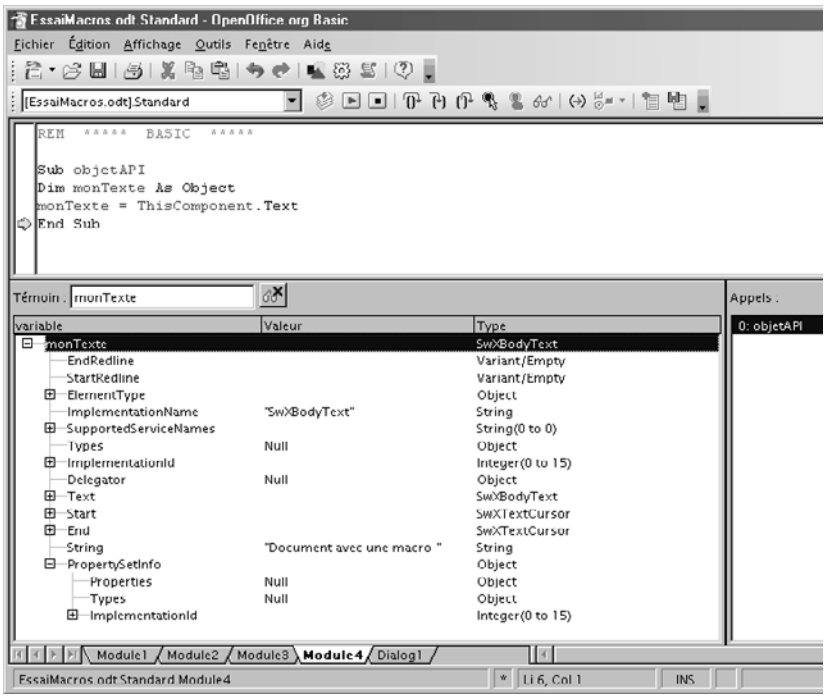


Figure 3–17
Afficher un objet de l'API

Nous verrons à l'annexe C que l'outil Xray permet d'obtenir encore plus d'information sur les objets API.

Poser des points d'arrêt

Maintenant, supposez que votre macro exécute 1 000 fois une boucle avant d'arriver à l'instruction qui vous intéresse. Impossible d'utiliser l'exécution instruction par instruction ! La solution est d'utiliser un point d'arrêt. Positionnez votre curseur sur la ligne comportant `MsgBox`. Cliquez sur l'icône Point d'arrêt (celle avec une grande main). Un rond rouge apparaît en marge de cette instruction. Cliquez sur l'icône Exécuter. La macro s'exécute et s'arrête avant d'exécuter l'instruction identifiée par le point d'arrêt. Le voyant rouge est allumé, vous pouvez lire la valeur de la variable. Cliquez de nouveau sur l'icône Exécuter : la macro reprend son déroulement jusqu'à la fin. C'est la méthode que nous avons utilisée pour la figure 3-17.

Vous pouvez poser plusieurs points d'arrêt dans un programme, dans plusieurs macros ou dans la même macro. Enlevez un point d'arrêt en répétant la méthode employée pour le poser. On peut aussi poser ou enlever un point d'arrêt par un double-clic en marge de l'instruction concernée.

Sur une exécution interrompue, lorsque vous déplacez la souris au-dessus d'une variable, une bulle affiche sa valeur.

Vérifier la syntaxe

La première icône de la deuxième barre d'outils s'appelle Compiler. En réalité, il n'est nul besoin de compiler un programme Basic avant de le lancer, car ceci est fait automatiquement avant toute exécution. Néanmoins, cette commande est utile pour effectuer une vérification de la syntaxe du module. Par exemple, supprimez la lettre `b` dans l'instruction `End Sub` de notre macro. Cliquez sur l'icône Compiler ; Basic ronchonne en vous affichant un message d'erreur. Si votre programme utilise des macros réparties dans plusieurs modules, effectuer une compilation à chaque modification d'un module vous évitera des erreurs bizarres et difficiles à trouver.

Modules et macros

On peut changer le nom d'un module en renommant son onglet ; soit par un clic droit sur l'onglet puis Renommer, soit par un Alt+clic sur l'onglet. L'éditeur Basic affiche les onglets dans l'ordre alphabétique, les modules d'abord, les boîtes de dialogue ensuite.

> **ATTENTION Nom de module**
>
> Ne donnez pas à un module le même nom que celui d'une macro de la même bibliothèque. Cela provoquerait des anomalies d'exécution.

Un clic droit sur la zone des onglets donne accès à l'insertion d'un nouveau module ou d'une nouvelle boîte de dialogue. Choisissez Module Basic. Vous obtenez un nouvel onglet intitulé `Module2` et, dans la zone d'édition, toujours la même macro vide :

```
REM  *****  BASIC  *****

Sub Main

End Sub
```

Comme les macros ne sont pas locales à un module mais communes à toute la bibliothèque, il est recommandé qu'elles aient des noms tous différents. Renommez donc systématiquement le sempiternel `Main`. Ceci n'est cependant pas imposé par OOoBasic.

Par un simple clic sur l'onglet, vous affichez instantanément un des modules de la bibliothèque. Répartir les macros dans différents modules facilite leur accès pour l'édition et pour l'exécution, car nous avons vu que les icônes Exécuter et Étape par étape lancent seulement la première macro du module.

Autres fonctionnalités de l'éditeur

Dans la deuxième barre d'outils, les icônes Macros et Modules ouvrent les panneaux que nous avons déjà décrits. Toujours dans cette barre, l'icône Enregistrer sous... écrit un fichier contenant le texte du module. Ce fichier possède l'extension `.bas`.

Sur un arrêt d'exécution, la zone Appels, en bas à droite de la fenêtre, affiche la pile des appels successifs de macros. En effet, un programme est décomposé en nombreuses macros, qui s'appellent l'une l'autre en cascade.

Une ligne d'instruction (ou de commentaire) peut être plus grande que la zone d'affichage. L'ascenseur horizontal se trouve en dessous de la zone Appels.

L'environnement de développement des autres langages

La version 2.0 d'OpenOffice.org fournit dans la section Macros OpenOffice.org des exemples de macros réalisant la même chose dans différents langages, en particulier écrire « Hello World ». C'est une occasion de jeter un œil sur ces langages alternatifs, qui ont chacun leurs spécificités. Nous ne nous attarderons pas dessus car le reste du livre utilisera le Basic.

LIMITATION Protection des scripts

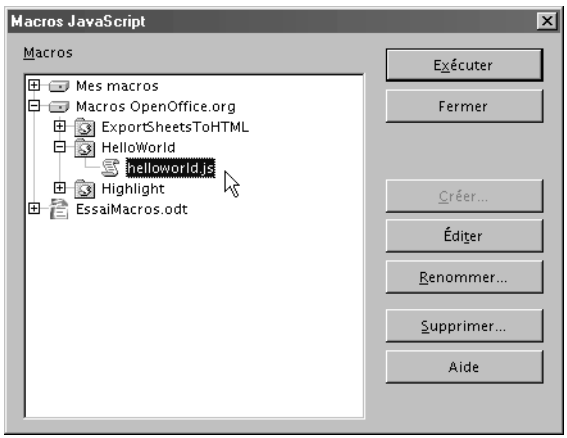
Seules les bibliothèques Basic peuvent être protégées par mot de passe. Les scripts réalisés dans d'autres langages sont toujours visibles en clair et modifiables (dans `Mes Macros` ou un document).

L'arborescence de menu que nous avons vue à la figure 3-5 peut nous mener aux différents panneaux de macros qui suivent.

JavaScript

JavaScript est un langage bien connu des créateurs de sites Web. Dans le panneau Macros de la figure 3-18, chaque macro JavaScript est contenue dans un fichier ayant l'extension `.js`.

Figure 3–18
Panneau Macros JavaScript



Vous pouvez créer un nouveau dossier ou une nouvelle macro dans le dossier sélectionné. Le bouton Éditer affiche la fenêtre de l'éditeur Rhino (voir figure 3-19). C'est un éditeur OpenSource écrit en Java. Il offre des possibilités comparables à celles de l'éditeur Basic, à l'exception de la coloration syntaxique et de l'aide en ligne sur les instructions.

Vous trouverez plus d'informations sur les sites Web suivants :

- site de Rhino, en anglais : http://www.mozilla.org/rhino/ ;
- un site français sur JavaScript : http://www.toutjavascript.com/.

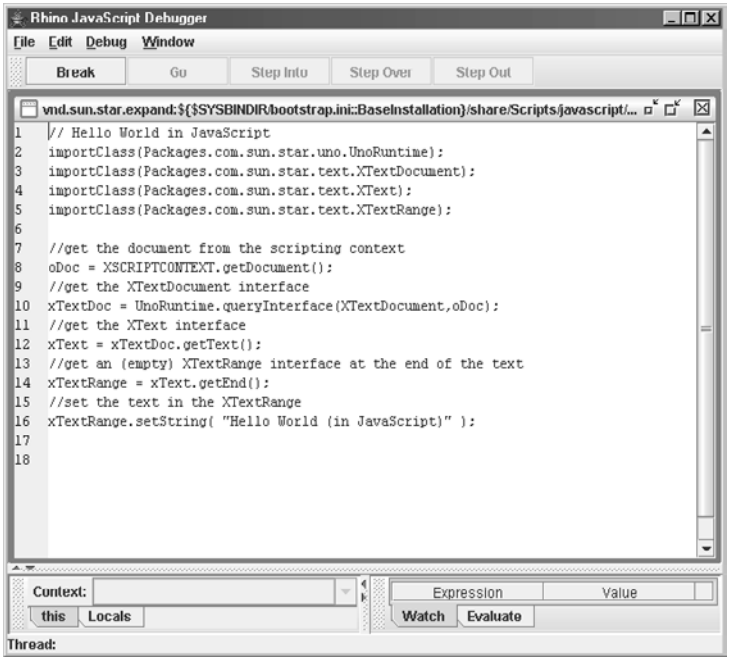


Figure 3–19
Editeur de JavaScript

BeanShell

BeanShell est une sorte de Java interprété et plus simple au niveau des déclarations. Dans le panneau **Macros** de la figure 3-20, chaque macro BeanShell est contenue dans un fichier ayant l'extension `.bsh`.

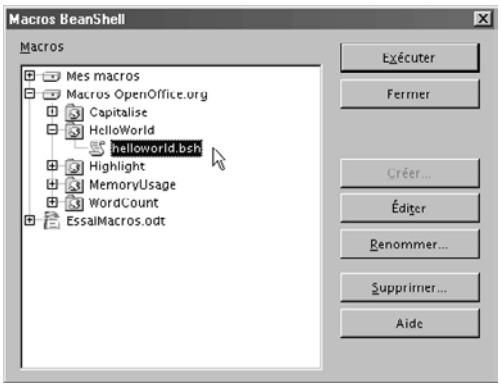


Figure 3–20
Panneau Macros BeanShell

Vous pouvez créer un nouveau dossier ou une nouvelle macro dans le dossier sélectionné. Le bouton **Éditer** affiche la fenêtre de l'éditeur BeanShell (voir figure 3-21). C'est un éditeur simple qui n'offre ni la coloration syntaxique, ni des moyens de mise au point.

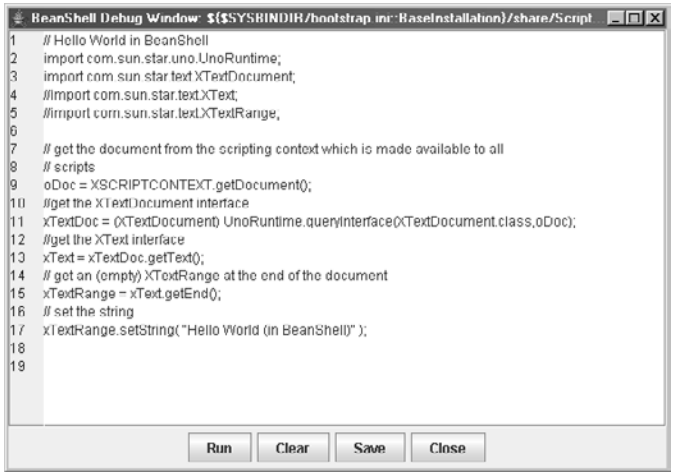


Figure 3–21
Éditeur de BeanShell

Pour plus d'informations sur BeanShell, consultez le site suivant : http://www.beanshell.org/.

Python

Python est un langage OpenSource très puissant, utilisant des concepts de programmation élégants et originaux, et qui pourtant peut donner un codage proche de Basic.

Dans le panneau Macros de la figure 3-22, le dossier `HelloWorld` est un fichier source `HelloWorld.py` et `HelloWorldPython` est le nom d'une fonction dans ce fichier. Un fichier source peut comporter plusieurs fonctions appelables.

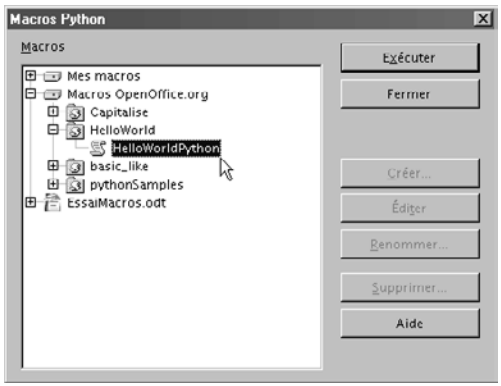


Figure 3–22
Panneau Macros Python

Vous remarquerez sur la figure 3-22 que le bouton Éditer est inactivé. Effectivement, OpenOffice.org n'offre pas encore d'éditeur pour Python. Il n'est pas non plus possible via l'interface utilisateur de créer un fichier Python ni de l'intégrer à OpenOffice.org. Le développeur de macros Python doit utiliser un éditeur externe et insérer son fichier manuellement dans l'installation.

Cette limitation est due au manque de développeurs ; il est à souhaiter que quelques-uns, ou des entreprises généreuses, proposent leur aide à la communauté OpenOffice.org afin de mieux intégrer ce langage.

Les programmeurs Python noteront qu'il est nécessaire d'utiliser l'interpréteur Python intégré à OpenOffice.org, et que l'IDE n'est pas non plus utilisable pour exécuter un script.

Voyez les sites suivants pour plus d'informations :

- le site Python, en anglais : http://www.python.org/ ;
- l'Association Francophone Python : http://www.afpy.org/.
- La page pyUNO, Python dans OpenOffice.org : http://udk.openoffice.org/python/python-bridge.html

Java compilé

Au chapitre 2, nous avons signalé qu'OpenOffice.org peut exécuter des scripts en Java compilé (fichiers `.jar`). Cependant, il n'existe pas de panneau Macros correspondant, et leur installation devra être faite manuellement par un programmeur confirmé. Ceci est décrit au chapitre *Scripting Framework* du *Developer's Guide*, qui fait partie du SDK, décrit en annexe A. Il s'agit bien entendu du SDK pour la version 2.0 d'OpenOffice.org.

Conclusion

OpenOffice.org fournit un environnement de développement complet permettant de gagner du temps dans la mise au point des macros Basic. Coloration syntaxique, débogage... sont autant d'outils indispensables pour créer ses macros dans de bonnes conditions. Les macros sont classées et hiérarchisées (en bibliothèques et modules), ce qui permet de rationaliser le développement.

La version 2.0 d'OpenOffice.org ouvre de nouvelles possibilités avec les scripts utilisant d'autres langages que Basic. Gageons que dans l'avenir leur prise en charge sera encore plus intégrée dans OpenOffice.org.

Après cette partie introductive, nous allons maintenant explorer dans la deuxième partie le langage OOoBasic lui-même.

Deuxième partie

Le langage OOoBasic

Il existe plusieurs langages appelés Basic. Ils se ressemblent, mais chacun a ses particularités. Même si vous connaissez déjà un Basic, par exemple Visual Basic ™ qui lui est proche, parcourez les chapitres de cette partie. En cas de problème d'exécution, relisez-la, elle contient bien des détails importants et pas toujours indiqués dans la documentation officielle. Nous vous donnons aussi quelques règles de bonne programmation, non pas pour vous enlever le plaisir de programmer, mais au contraire pour vous éviter échecs et découragements.

4

Introduction à la programmation Basic

Avec ce chapitre, vous débutez l'apprentissage du Basic OpenOffice.org. Vous ferez connaissance avec les règles de syntaxe du langage et nous vous donnerons quelques conseils de bonne programmation.

> **REMARQUE** **Débutants**
>
> Ne soyez pas rebuté par les termes employés dans les lignes de code reproduites ici. Ils seront expliqués progressivement dans les chapitres suivants. Pour le moment, écrivez-les exactement comme écrit.

Les éléments de base

Nous vous recommandons de poser votre livre sur la table de votre ordinateur et d'ouvrir l'EDI sur un nouveau module, comme vous l'avez appris dans le chapitre précédent. N'hésitez pas à mettre en application immédiate ce que nous allons exposer.

Vous avez donc probablement devant vous une page Basic écrite ainsi :

```
REM ***** BASIC *****

Sub Main

End Sub

Sub Macro1

End Sub
```

Comme nous l'avons déjà vu, cette page contient :

- une ligne de commentaire, commençant par `REM` ;
- une première macro nommée `Main` ;
- une deuxième macro nommée `Macro1`.

Cette description n'est pas complète. Nous avons en plus des lignes vides (ou lignes blanches) et des espaces entre les mots-clés.

Premier point important : un programme Basic est composé de *lignes*. Cela peut paraître une évidence sans intérêt, mais cette phrase veut dire que Basic utilise la découpe en lignes pour « comprendre » ce qu'on lui demande de faire. Ainsi, les termes `Sub Main` et `End Sub` doivent être sur deux lignes différentes et dans cet ordre.

Les éléments non interprétés

Certaines parties du codage ne sont jamais utilisées par Basic. Elles servent seulement à un lecteur humain.

Les espaces et tabulations

Basic permet d'employer un nombre quelconque d'espaces ou de tabulations sur chaque ligne de codage. Ceci n'aura pas d'influence sur la vitesse d'exécution des macros. Basic ne considère ces éléments que comme séparateurs entre deux mots, ou à l'intérieur d'une chaîne de caractères. Ceci lui est donc tout à fait équivalent :

```
     REM                          ***** BASIC *****

Sub            Main


         End                     Sub
  Sub Macro1
  End Sub
```

Contrairement à cet exemple, en utilisant judicieusement des espaces, vous pouvez simplifier considérablement la relecture de vos programmes en mettant en évidence leur structure.

Les commentaires

Les commentaires débutent par le mot-clé REM. À partir de ce mot, Basic ignore complètement le reste de la ligne. Vous pouvez y mettre un texte quelconque, y compris avec des caractères accentués. Le caractère apostrophe ' (la touche 4 sur le clavier principal) remplit la même fonction que le mot-clé REM. Voici plusieurs exemples de commentaires, que nous reproduisons en gris clair dans ce livre :

```
REM ***** BASIC *****
rem Code04-01.sxw   bibli : Standard Module1
Sub Main ' une macro appelée Main
  'MsgBox "Bonjour"
End Sub REM fin de la macro Main
```

Basic ne distingue pas les majuscules des minuscules pour les mots-clés et les noms de variables. Ainsi nous aurions pu employer Rem pour nos commentaires.

> **ASTUCE Les exemples du livre**
>
> Les exemples complets du livre sont déjà écrits dans des documents Writer, Calc, Draw que vous trouverez dans le zip téléchargeable accompagnant ce livre. Copiez le document sur votre disque dur et ouvrez dans l'EDI le module de la bibliothèque indiquée dans la ligne `rem Codexx-yy`.

Longueur des lignes

Basic accepte les lignes de très grande longueur. Néanmoins, comme l'éditeur n'affiche qu'une zone limitée, faites attention aux mots qui pourraient se cacher derrière une séquence d'espaces ou tabulations.

Un livre ne peut malheureusement pas représenter de longues lignes de codage. Aussi serons-nous contraints de couper certaines lignes trop longues. Nous emploierons alors le caractère « souligné » _ (la touche 8 du clavier principal), qui est reconnu par Basic comme indiquant que la même instruction se poursuit sur la ligne suivante. Ce caractère doit être le dernier de la ligne, on ne peut mettre ni espace ni commentaire à sa droite ; il ne doit pas couper un mot ni être accolé à lui.

```
rem Code04-01.sxw   bibli : Standard Module1

Sub Macro1 ' un peu de poésie dans ce monde informatique
MsgBox("Là tout n'est qu'ordre et beauté," & chr(13) & _
       "Luxe, calme et volupté."           & chr(13) & _
       "             Charles Beaudelaire")
End Sub
```

> **À NOTER Commentaires et chaînes de caractères**
>
> Le seul moyen de continuer un commentaire sur plusieurs lignes est de répéter le `REM` ou l'apostrophe à chaque ligne. Le caractère de continuation est traité comme un caractère ordinaire à l'intérieur d'une chaîne de caractères.

Instructions Basic

Les instructions Basic simples s'écrivent sur une seule ligne. Il est possible de mettre plusieurs instructions Basic sur la même ligne en les séparant par le caractère deux points « : ». Toutefois, évitez ce style d'écriture car vous auriez plus de difficulté à relire vos programmes.

D'autres instructions s'écrivent obligatoirement sur plusieurs lignes, comme `Sub ... End Sub`.

Basic est un langage de programmation de langue anglaise. Les mots-clés qui le composent sont des mots anglais (même si certains sont aussi des mots français comme `Date`, `Environ`, `Minute`). Il en est de même des noms employés par l'API, que nous utiliserons à partir du chapitre 10.

Noms des variables et des routines

Un programmeur utilise de nombreuses routines (sous-programmes), variables ou constantes. Il les désigne chacune par un nom particulier.

Basic est indifférent à la *casse* des caractères, c'est-à-dire qu'il ne distingue pas un caractère alphabétique en minuscule du même caractère en majuscule (nous nuancerons cependant ceci quand nous aborderons l'utilisation des constantes nommées de l'API). Les noms doivent de plus respecter les règles suivantes :

- comporter entre 1 et 255 caractères ;
- avoir un premier caractère alphabétique (une des 26 lettres de l'alphabet, majuscule ou minuscule mais non accentuée) ;
- ou bien avoir pour premier caractère le souligné _ à condition qu'il soit suivi d'au moins un autre caractère alphabétique, numérique (0 à 9) ou souligné ;
- avoir des caractères suivants alphabétiques, numériques ou le caractère souligné ; évitez de terminer le nom par un souligné.

Basic accepte, sous certaines conditions précisées dans l'aide en ligne, des caractères espace dans un nom. Nous déconseillons cet usage.

Les caractères alphabétiques accentués ou les caractères spéciaux sont interdits dans un nom. Cette limitation sera partiellement levée dans une version future. Certains caractères comme $ et % sont utilisés dans les noms, mais pour des usages particuliers que nous expliquerons au chapitre 5.

Syntaxe élémentaire

L'affectation d'une valeur à une variable s'écrit simplement avec le signe égal :

```
unCompteur = 3
```

La partie à droite du signe égal peut comporter un simple nombre ou une autre variable, mais peut aussi bien être une expression assez complexe :

```
unCompteur = unCompteur + comptesClient(numClient)
```

Les macros et beaucoup d'instructions Basic peuvent comporter des paramètres en argument, ou aucun argument. En principe, les arguments sont encadrés de parenthèses comme ici :

```
MsgBox("Le travail est terminé", messInfo, titreFenetre)
```

Basic accepte, dans une instruction aussi simple, d'omettre les parenthèses :

```
MsgBox "Le travail est terminé", messInfo, titreFenetre
```

En revanche, si la macro ou instruction est un élément d'une expression plus élaborée, les parenthèses sont nécessaires. En cas de doute, utilisez-les.

Dans le cas particulier d'une macro ou instruction Basic n'utilisant aucun argument, les deux formes suivantes sont donc possibles :

```
macro1()

macro1
```

Ces deux formes sont autorisées dans toutes les constructions.

> **ATTENTION Parenthèses**
>
> Les parenthèses jouent un rôle différent pour les variables. Une variable change de signification avec ou sans parenthèses. Voir le chapitre 5.

Certaines macros et instructions Basic peuvent renvoyer une valeur résultat ; on les appelle des fonctions. Le résultat de la fonction peut être volontairement ignoré.

```
rem Code04-01.sxw   bibli : Standard Module2
Sub Main2
print Time ' la valeur renvoyée est affichée
Time ' la valeur renvoyée est ignorée
End Sub
```

En fait, ici la deuxième ligne ne sert à rien. Cela pourrait être une erreur de programmation, et Basic ne produira aucun message d'erreur.

Exécution Basic

En principe, vous ne pouvez réaliser qu'une seule exécution Basic à la fois. Avec un écran d'ordinateur submergé de fenêtres, on peut avoir oublié qu'un programme Basic est en cours d'exécution, stoppé à un point d'arrêt. Or, beaucoup de petits ajouts bien pratiques sont réalisés avec des macros Basic. Le déclenchement d'une macro Basic à partir d'une icône de barre d'outils dans un document OpenOffice.org peut alors donner des résultats bizarres.

Tant que Basic est en cours d'exécution, OpenOffice.org ne vous permet pas de *fermer* le document qui a lancé la macro.

Recommandations de programmation

Comme indiqué dans ce titre, nous vous présentons ici des recommandations. Vous n'êtes pas obligé de les suivre, ni d'être d'accord. Elles sont cependant acceptées par bien des professionnels.

Certaines des recommandations livrées ici concernent des concepts que nous aborderons plus loin dans ce livre. Aussi n'hésitez pas à les relire de temps en temps. Toutes ont le même but : vous aider à écrire des programmes clairs, maintenables (c'est-à-dire qui peuvent être modifiés longtemps après, même par une autre personne), et moins sujets aux erreurs.

Le choix des mots

Entre 1 et 255 caractères, il y a place pour des noms de variables et de routines d'une longueur raisonnable. Le programme ne sera pas ralenti avec des noms de grande longueur. En revanche, il sera plus difficile à relire et à mettre au point si les noms sont trop courts ou trop longs.

Dans la mesure où vous n'êtes pas contraint par des règles de programmation propres à votre entreprise, choisissez pour vos noms de variables et macros des noms significatifs de leur utilité, et écrits en minuscules, sauf pour mieux repérer des mots lorsque ce nom est une concaténation de mots. Exemples de variables :

```
monDocument, numeroClient, couleurCadre
```

Donnez à vos macros `Sub` un nom comportant un verbe à l'infinitif :

```
imprimer()
```

Pour une macro renvoyant une valeur booléenne (vrai, faux) choisissez un nom qui se combine avec l'instruction `if` de manière significative :

```
if clientExiste(nomClient) then
```

En employant de tels noms, votre codage deviendra « auto-documenté » c'est-à-dire qu'à la lecture des instructions, vous aurez déjà une bonne idée du traitement réalisé. Ceci ne vous dispense pour autant pas de commenter les séquences complexes.

Pour éviter des erreurs sur le nom des variables et routines, utilisez systématiquement le copier-coller (ou Ctrl + glisser-déplacer avec la souris).

Inversement, pour des variables « de travail » à usage très localisé, par exemple un simple index dans une boucle, il est plus lisible d'utiliser une variable à un ou deux caractère(s). Évitez les variables à un caractère dont la graphie ressemble à un chiffre (lettres i, l, o). Préférez les variables comme x, x2, n, p3, aa.

Certains d'entre vous peuvent avoir besoin de programmer en langue anglaise. Méfiez-vous des noms de variables anglais, vous risquez des collisions avec des noms réservés ou des noms utilisés par l'API OpenOffice.org. Un bon moyen d'éviter ces problèmes est de mettre un ou plusieurs chiffre(s) dans le nom.

N'hésitez pas non plus à changer le nom d'une variable ou d'une macro si cela peut clarifier le codage. L'EDI vous permet de faire un chercher-remplacer sur une zone sélectionnée aussi bien que dans tous les modules de la bibliothèque.

Les noms de modules devraient aussi être significatifs, mais des noms longs rendent les onglets difficiles à manipuler. Ne choisissez pas comme nom de module le nom d'une des macros de celle-ci.

Choisissez bien le nom de vos bibliothèques, afin d'éviter que ce nom soit susceptible d'être aussi choisi par un autre programmeur pour sa propre bibliothèque.

La forme : commentaires, espaces

Utilisez les lignes vides et les espaces (ou les tabulations) pour aérer votre codage et mettre en valeur certaines structures. Cela ne prend pas beaucoup plus de temps à écrire et vous aidera grandement dans vos relectures.

Ajoutez des commentaires, là où c'est utile, avec un texte clair et concis. Supposez que votre programme doive être modifié par un autre programmeur de compétence moyenne ; ou que vous ayez à le modifier cinq ans après l'avoir terminé. Ne commentez cependant pas les codages simples et évidents, voyez ce qui est dit à propos du choix des noms.

Modularité

Découpez votre programme en multiples macros. Idéalement, chaque macro devrait être entièrement visible dans la fenêtre d'édition de l'EDI. Ceci implique d'appeler des macros en cascade. Chaque macro devrait réaliser une tâche clairement délimitée.

Évitez les variables globales (voir chapitre 5) et préférez les variables locales à une macro ; transférez les informations dans des paramètres de macro.

Autant que possible, testez chaque macro séparément, en commençant par celles qui n'en appellent aucune autre. Si vous ne comprenez pas pourquoi votre macro fonctionne mal, simplifiez-la jusqu'à en trouver la raison.

On peut regrouper dans un même module les macros concourant à la réalisation d'une tâche complexe. La découpe en plusieurs modules permet de passer facilement d'une partie à une autre du programme.

Servez-vous des codages déjà réalisés comme base pour vos nouvelles macros. Utilisez abondamment le copier-coller puis adaptez à vos besoins.

Ne ré-inventez pas la roue à chaque nouveau programme ! Avec l'expérience et en vous basant sur des codages existants, vous deviendrez capables de créer des macros génériques qui, une fois écrites et testées, seront réutilisables sans modifications et pourront être considérées comme des super-instructions Basic. Cette approche, si vous développez beaucoup de macros, vous fera gagner un temps précieux et vous pourrez ainsi vous concentrer sur la finalité de la macro que vous êtes en train de développer.

Simplicité

Il y a en général une infinité de manières de résoudre un problème par un programme. Préférez les solutions simples et robustes aux solutions astucieuses mais peu évidentes. Reprenez plusieurs fois votre codage, car la première solution est rarement une solution simple.

La simplicité résulte aussi d'une parfaite compréhension du sujet à traiter, et d'une bonne connaissance des possibilités de votre langage de programmation. Vous ne les acquerrez qu'avec le temps et la pratique.

Robustesse

Évitez de mettre un nombre ou une chaîne de caractère directement dans une instruction, si ce nombre ou cette chaîne est employé plusieurs fois ou est susceptible d'évoluer dans le futur, ou a une signification particulière. Utilisez une constante ou une variable à la place et initialisez-la à un endroit bien visible :

```
rem Code04-01.sxw   bibli : Standard Module3
Sub Main3
Const titreFenetre = "Ma jolie macro"
Const messInfo = 64 ' valeur indiquant le type du message
' - - -
' - - - ici les instructions de la macro - - -
' - - -
MsgBox("Le travail est terminé", messInfo, titreFenetre)
End Sub
```

Chaque fois que vous devez recevoir une valeur provenant de l'utilisateur, ou de la lecture d'un fichier, attendez-vous au pire. C'est-à-dire, avant de commencer tout traitement, vérifiez par votre codage que la donnée obtenue est bien dans la gamme des valeurs autorisées. Si plusieurs données doivent être recueillies, vérifiez la cohérence entre elles.

Relectures

L'ordinateur n'a aucune intelligence ; il ne fait que ce que vous lui dites de faire, et il le fait aveuglément. La moindre erreur peut rendre votre superbe construction totalement inutilisable. Or, les êtres humains sont faillibles, et malgré toute la concentration que vous apporterez, malgré votre expérience, vous ferez des erreurs, parfois difficiles à trouver.

Ne vous dites pas : je mettrai tout ça au point avec les tests. En réalité, aucune série de tests ne peut garantir le bon fonctionnement d'un programme un peu complexe. Elle peut seulement démontrer que le programme ne marche pas. De plus, définir un ensemble de tests efficace est un travail aussi complexe que d'écrire un programme.

N'hésitez donc pas à relire et relire encore vos codages de manière critique. Avancez pas à pas dans votre lecture en vous posant des questions comme : la variable est-elle bien initialisée ? L'appel de cette instruction est-il correct ? Ai-je bien vérifié toutes les branches d'exécution de cette macro ? Ce codage ne pourrait-il pas être simplifié ?

Conclusion

Nous venons de présenter le cadre syntaxique général de OOoBasic. Les conseils qui ont été donnés en termes d'habitudes d'écriture permettront d'écrire des macros claires et documentées. Ce ne pourra être que bénéfique par la suite.

Le chapitre suivant traite des variables, briques de bases essentielles à la programmation.

5

Variables et tableaux de variables

Qu'est-ce qu'une variable ? C'est une zone de mémoire volatile à laquelle on donne un nom, et qui sert à mémoriser une information. Mémoire volatile veut dire que celle-ci ne se conserve pas si on coupe l'alimentation de l'ordinateur. Basic dispose de différents types de variables, pour différents usages. Nous allons les décrire dans ce chapitre.

Déclarer des variables

Le programmeur déclare une variable pour prévenir l'ordinateur qu'il doit réserver une zone mémoire et lui donner un nom.

OpenOffice.org Basic n'oblige pas à déclarer les variables simples. Ainsi, le code suivant utilise une variable nommée `Chanson`.

```
rem Code05-01.sxw   bibli : Library1 Module1
Sub SansDeclaration()
Chanson = "Au clair de la Lune"
print Chanson
End Sub
```

Ceci est très pratique, mais à déconseiller car une simple faute de frappe peut créer involontairement des variables.

Le code suivant en montre une conséquence :

```
rem Code05-01.sxw   bibli : Library1 Module2
Sub ErreurFrappe()
Chanson = "Au clair de la Lune"
' - - - -
' supposons toute une série d'instructions ici
' - - - -
Chason = "Il pleut Bergère"
' - - - -
' supposons toute une série d'instructions ici
' - - - -
print Chanson
End Sub
```

En exécutant cette macro vous verrez le message : « Au clair de la Lune », au lieu de « Il pleut Bergère ».

Vous avez fait une petite faute de frappe et votre programme fonctionne d'une manière inattendue, mais sans erreur. Sachant que dans des applications complexes, vous pouvez avoir besoin de plusieurs dizaines de variables utilisées chacune plusieurs fois, mesurez les risques que vous prenez.

La déclaration Explicit

Pour éviter ces erreurs, Basic dispose d'une déclaration optionnelle :

```
Option Explicit
```

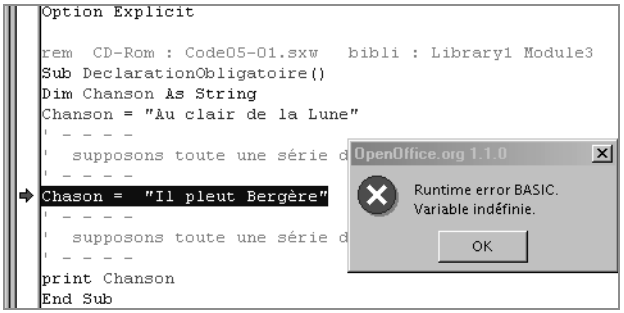
Cette ligne doit être écrite au début du module, avant toute instruction. Elle peut être précédée de commentaires, d'espaces, ou de lignes blanches. Après cette déclaration, toute variable utilisée doit être déclarée par une instruction `Dim` avant de l'utiliser ou de l'initialiser. Modifions le code précédent, qui devient :

```
rem Code05-01.sxw   bibli : Library1 Module3
Option Explicit

Sub DeclarationObligatoire()
Dim Chanson As String
Chanson = "Au clair de la Lune"
' - - - -
' supposons toute une série d'instructions ici
' - - - -
Chason = "Il pleut Bergère"
' - - - -
' supposons toute une série d'instructions ici
' - - - -
print Chanson
End Sub
```

Si vous exécutez cette macro, Basic affichera un message d'erreur (figure 5-1).

Figure 5–1
Erreur d'exécution



Non seulement Basic vous prévient d'une anomalie, mais il ouvre l'EDI si nécessaire et sélectionne la ligne en faute ! Vous obtiendriez le même message si l'erreur de frappe se situait non ici, mais à l'instruction `print`.

Si la déclaration `Dim` n'existait pas, Basic afficherait une erreur dès la première instruction d'affectation. En effet, `Dim` sert à déclarer à Basic les variables que nous voulons utiliser.

La déclaration Dim

Nous venons de voir une forme très courante de la déclaration `Dim`.

```
Dim Chanson As String
```

Elle signifie : je déclare que la variable `Chanson` est du type `String`. Nous verrons plus bas ce qu'est le type `String`. Le mot réservé `As` doit précéder le type de la variable.

Il existe une forme plus simple :

```
Dim Chanson
```

Pour Basic, cette forme est équivalente à :

```
Dim Chanson As Variant
```

`Variant` est un autre type de variable, utilisé par défaut, que nous décrirons plus loin. Il est nettement préférable de toujours déclarer explicitement le type `Variant` (et de l'employer seulement à bon escient).

On peut, soit déclarer chaque variable avec une déclaration `Dim`, soit les regrouper dans une même déclaration :

```
Dim Chanson As String, Chanteur As String, Annee As Integer
```

Même si les variables sont du même type, vous devez répéter le `As ...` pour chacune. En effet, l'instruction suivante est acceptée mais signifie autre chose :

```
Dim Chanson, Chanteur As String, Annee As Integer
```

Elle signifie que la variable `Chanson` est du type `Variant` puisque vous n'avez pas précisé le type. Il existe d'autres formes de déclarations `Dim`, employées pour les tableaux ; elles sont décrites plus loin dans une section spécifique.

Valeur initiale des variables

Basic donne une valeur initiale à chacune de vos variables. La valeur exacte dépend du type de la variable ; par exemple, elle équivaut à zéro pour les variables numériques.

Vous ne devriez pas compter sur Basic pour initialiser vos variables ; prenez plutôt la peine de le faire vous-même. Vous déciderez ainsi à quel endroit le faire (parfois il faut réinitialiser en cours de traitement), et quelle valeur donner (parfois une valeur autre que zéro est préférable).

L'intérêt de l'initialisation des variables par Basic est que, si vous avez oublié d'initialiser une variable, votre programme se comportera de manière reproductible, ce qui facilite la recherche des erreurs de programmation. Alors que dans d'autres langages, une variable non initialisée par le programmeur peut dépendre des traces laissées en mémoire par les applications précédemment exécutées.

Portée des variables

Les variables déclarées dans une routine `Sub` ou `Function` sont connues seulement dans cette routine. Plus exactement, les variables sont connues à l'intérieur de la routine à partir du moment où elles sont déclarées. Il n'est pas obligatoire de regrouper toutes les déclarations en début de routine, il suffit de déclarer la variable juste avant la première utilisation. Néanmoins, les déclarations au fil de l'eau aboutissent à un code moins lisible que des déclarations regroupées en un ou deux endroit(s).

Puisque les variables d'une routine ne sont pas connues à l'extérieur de celle-ci, il est possible de reprendre les mêmes noms dans différentes routines.

À l'inverse, il est possible de définir des variables communes à plusieurs routines. Les programmeurs expérimentés évitent cela autant que possible car la mise au point est rendue plus délicate, mais c'est parfois absolument nécessaire. Plusieurs cas se présentent.

Variable commune à un module

Dans le code qui suit, la routine principale `Main1()` affecte une valeur à la variable `Chanson`, appelle la routine `ModifierChanson()` et affiche le contenu de la variable.

```
rem Code05-01.sxw   bibli : Library2 Module1
Option Explicit

Private Chanson As String

Sub Main1()
CompatibilityMode(True) ' nécessaire pour que Private fonctionne
Chanson = "Au clair de la Lune"
ModifierChanson()
print Chanson
End Sub

Sub ModifierChanson()
Chanson = "Il pleut Bergère"
End Sub
```

La déclaration `Dim` est remplacée par une déclaration `Private`, en dehors des deux routines mais dans le même module. `Private` s'emploie avec la même syntaxe que `Dim`.

De plus, nous avons ajouté l'instruction `CompatibilityMode(True)` au début de la macro principale. Cette instruction permet le fonctionnement correct de `Private` ; en son absence, la variable `Chanson` resterait connue dans tous les modules de la bibliothèque. Ce mode de compatibilité a été introduit pour corriger une bogue tout en évitant que d'anciennes macros ne fonctionnent plus ; il doit être activé dynamiquement (en exécutant l'instruction avant d'utiliser la variable) et il peut être désactivé de la même manière en exécutant `CompatibilityMode(False)`. Il n'est pas possible de tester l'état courant de `CompatibilityMode`.

L'exécution de la routine `Main1()` montre que la variable est bien accessible depuis les deux routines.

BOGUE La déclaration Private

Si la déclaration `Private` est employée seule, elle n'est pas effective dans OpenOffice.org. Les variables ainsi déclarées sont connues dans tous les modules de la bibliothèque. L'instruction `CompatibilityMode` permettant de corriger ce comportement n'est disponible qu'à partir de la version 1.1.1.

Variable commune à une bibliothèque

Nous allons déclarer une variable dans un module et l'utiliser dans un autre module de la même bibliothèque. Voici le module déclarant une variable commune :

```
rem Code05-01.sxw   bibli : Library3 Module2
Option Explicit

Public Chanson As String

Sub ModifierChanson()
```

```
Chanson = "Il pleut Bergère"
End Sub
```

La déclaration `Public` s'emploie avec la même syntaxe que la déclaration `Dim`.

Dans un autre module de la même bibliothèque, nous plaçons la routine principale, qui affecte une valeur à la variable commune, appelle une routine qui se trouve dans l'autre module, et affiche le contenu de la variable :

```
rem Code05-01.sxw   bibli : Library3 Module1
Option Explicit

Sub Main1bis()
Chanson = "Au clair de la Lune"
ModifierChanson()
print Chanson
End Sub
```

Variable commune à plusieurs bibliothèques

Nous allons déclarer une variable dans un module d'une bibliothèque.

```
rem Code05-02.sxw   bibli : Library1 Module1
Option Explicit

Global Compositeur As String, Naissance As Integer

Sub AffecterGlobal()
Compositeur = "Vivaldi"
Naissance = 1678
End Sub
```

La déclaration `Global` s'emploie avec la même syntaxe que la déclaration `Dim`.

Dans *une autre* bibliothèque du même document, utilisons cette variable :

```
rem Code05-02.sxw   bibli : Library2 Module1
Option Explicit

Sub UtiliserGlobal()
print Compositeur, Naissance
End Sub
```

À condition que la bibliothèque `Library1` soit chargée, l'exécution de `UtiliserGlobal()` se fera sans erreur. Si `AffecterGlobal()` a été exécutée auparavant, le résultat affiché sera « Vivaldi 1678». La variable ainsi définie n'est connue que dans les bibliothèques du même document. Si elle est définie dans une des bibliothèques de `soffice`, elle sera connue dans toutes les autres bibliothèques de ce conteneur.

Dans la bibliothèque où une variable `Global` est déclarée, la déclaration est équivalente à `Public`, c'est-à-dire qu'elle est connue de tous les modules.

PIÈGE Les variables Global sont persistantes

Quand la macro se termine, les variables `Global` mémorisent la dernière valeur affectée lors du traitement. Si vous relancez la macro, ou une autre macro les utilisant, ne comptez pas sur Basic pour les réinitialiser. Vous devez en tenir compte dans vos traitements. La mémorisation disparaît si le document contenant la déclaration de ces variables est fermé.

Les chaînes de caractères

Les variables de type `String` sont appelées chaînes de caractères. Elles contiennent une série de 0 à 65 535 caractères, appelée plus communément texte. La chaîne de caractères est construite en encadrant le texte de guillemets :

```
Chanson = "Au clair de la Lune"
```

Tous les espaces et la casse sont significatifs à l'intérieur de la séquence encadrée par les guillemets.

Une chaîne avec aucun caractère est une chaîne de longueur nulle, qui s'écrit ainsi :

```
Chanson = ""
```

Basic initialise les variables `String` à une chaîne de longueur nulle.

Une chaîne de caractères peut comporter des caractères guillemets. Chacun de ceux-ci doit alors être dupliqué. L'EDI facilite l'écriture de telles chaînes grâce à sa coloration syntaxique.

```
a = "Je vous dit: ""Bonjour!"""
```

L'opérateur de concaténation & sert à fusionner deux chaînes de caractères.

```
rem Code05-03.sxw   bibli : Library1 Module2
Option Explicit

Sub Abouter()
Dim Texte1 As String, Texte2 As String
Dim Resultat As String

Texte1 = "Le temps"
Texte2 = "beau"
Resultat = Texte1 & " est " & Texte2
print Resultat
```

```
Resultat = Resultat & " et chaud"
print Resultat
End Sub
```

L'opérateur + est aussi accepté à la place de l'opérateur & mais ce dernier est préférable pour éviter les ambiguïtés de code, notamment lors de l'utilisation du type `variant`. De nombreuses autres opérations sont possibles avec les chaînes de caractères, en utilisant les fonctions intégrées de Basic décrites au chapitre 8.

Une variable `String` est constituée de caractères Unicode à 16 bits. De nombreux caractères nationaux sont représentables.

POUR LES EXPERTS Le type Caractère

OpenOffice.org Basic ne possède pas de type Caractère. On mémorise un caractère par une chaîne à un caractère.

Les variables numériques

OpenOffice.org Basic dispose de plusieurs types de données pour représenter des nombres. Il est nécessaire de connaître leurs spécificités pour bien les utiliser.

Les entiers

Il existe deux types de variables numériques pouvant contenir des nombres entiers, voir le tableau 5-1.

Tableau 5–1 Les types entiers

Type	Gamme des valeurs	Occupation mémoire
`Integer`	-32 768 à +32 767	2 octets = 16 bits
`Long`	-2 147 483 648 à +2 147 483 647	4 octets = 32 bits

Pourquoi ces bornes bizarres ? Parce que les variables sont stockées sous forme binaire.

Pourquoi deux types ? Historiquement, ils étaient justifiés par l'occupation mémoire, deux fois plus grande pour le type `Long`, et des calculs plus longs pour ce type. Aujourd'hui, les capacités mémoire et la vitesse des processeurs actuels rendent ces critères obsolètes. On peut donc n'utiliser que le type `Long`, c'est moins long à écrire et ça évite les problèmes liés à des valeurs un peu grandes.

En Basic, les nombres entiers (et les autres nombres) sont écrits sans employer de caractère d'espacement de milliers.

```
Dim grand As Long
grand = 123456789
```

Basic initialise les variables `Integer` et `Long` à la valeur zéro.

Les opérateurs mathématiques pour les variables entières sont :

- `+` addition
- `-` soustraction
- `*` multiplication
- `/` division arrondie
- `\` division entière
- `mod` reste de la division entière

Cet exemple montre la différence entre division arrondie et division entière.

```
rem Code05-03.sxw   bibli : Library1 Module1
Option Explicit

Sub DivEntiers()
Dim a As Long, b As Long, c As Long, d As Long

a = 57
b = 6
c = a/b ' le résultat de la division est arrondi
d = a\b ' division entière
print c, d
End Sub
```

L'exécution de la routine donne c=10 et d=9.

Les nombres réels

Il existe deux types de variables numériques pouvant contenir des nombres à virgule flottante, voir le tableau 5-2.

Tableau 5–2 Les types réels

Type	Gamme des valeurs non-nulles	Précision	Occupation mémoire
`Single`	± 3,402823 x 10^{38} à ±1,401298 x 10^{-45}	6 à 7 chiffres	4 octets = 32 bits
`Double`	±1,79769313486232 10^{308} à ± 4,94065645841247 10^{-324}	14 à 15 chiffres	8 octets = 64 bits

Ces types sont des approximations des nombres réels, avec une précision plus grande pour le type `Double`.

Pourquoi deux types ? Historiquement, ils étaient justifiés par le coût de la mémoire et la lenteur des calculs en `Double`. Aujourd'hui, ces critères sont obsolètes pour la plupart des applications. On peut donc n'utiliser que le type `Double`.

Les nombres en virgule flottante sont écrits en Basic sans séparateur de milliers, avec un point pour marquer la décimale, et la lettre « E » ou « e » pour marquer le début de l'exposant :

```
Dim e As Double, h As Double
e = 2.718281828459045
h = 6.625e-34
```

Notez que, dans la troisième ligne, le « e » annonce l'exposant ; ce n'est pas la variable e ! Cet exposant note une puissance de 10, il n'a rien à voir avec l'exponentielle.

L'exposant est obligatoirement un entier. Vous devez utiliser le symbole puissance `^` pour effectuer des calculs comme les racines « nièmes » :

```
v = 7^0.5    ' racine carrée de 7
v = 27^(1/3) ' racine cubique de 27
```

Basic initialise les variables `Single` et `Double` à la valeur zéro.

Les variables en virgule flottante utilisent les opérateurs mathématiques suivants :

`+`	addition
`-`	soustraction
`*`	multiplication
`/`	division
`mod`	reste de la division entière
`^`	élévation à la puissance

Basic fournit des fonctions numériques et trigonométriques pour les calculs en virgule flottante. Elles sont listées au chapitre 8.

Les inconvénients du calcul en flottant

Les variables réelles sont stockées sous forme binaire flottant. Nous écrivons et lisons les nombres réels dans une représentation décimale flottante, et l'ordinateur doit donc faire une conversion pour les mémoriser et réaliser ses calculs. Cette conversion peut entraîner des petites erreurs, qui se propagent et apparaissent parfois visiblement dans le résultat. À cause de ces limitations, on ne doit pas tester l'égalité de deux variables à virgule flottante, mais tester si la valeur absolue de la différence est suffisamment faible. Le codage suivant présente un cas typique :

```
rem Code05-03.sxw   bibli : Library2 Module1
Option Explicit

Sub CalculsFlottants()
Dim A As Double, B As Double, C As Double
Dim R1 As Double, R2 As Double

A = -1.2345
B = 1.2346
C = -0.0001
R1 = B + A + C
R2 = B + C + A
print "R1=" & R1 ' résultat non nul
print "R2=" & R2 ' résultat nul
End Sub
```

Les deux nombres `A` et `B` sont voisins mais nettement différents si on tient compte de la grande précision du type `Double`. Cependant, l'exécution de la macro donne :

- `R1 = -1,10182045778839 10-17`
- `R2 = 0`

Le pur mathématicien est scandalisé : d'abord, il y a une erreur de calcul, et puis Basic ne connaît même pas la propriété de commutativité !

Ce phénomène se retrouve dans tous les ordinateurs calculant en binaire flottant. L'erreur apparaît quand on fait la différence entre deux nombres voisins et qu'on utilise le résultat sans précaution.

Les variables monétaires

Nous avons vu avec les nombres réels que l'implémentation est entachée d'une imprécision, faible mais inévitable. Les comptables des grandes entreprises, les employés de banque, les fonctionnaires de l'État, sont amenés à manipuler des nombres de très grande valeur avec plusieurs décimales (en général 2, parfois plus) avec une exigence absolue de précision.

Pour ces usages, Basic fournit le type monétaire appelé `Currency`. Les variables monétaires sont implémentées sous la forme de nombres entiers de 8 octets, soit 64 bits. L'affichage d'une variable monétaire considère que les 4 derniers chiffres de la représentation décimale sont quatre décimales du nombre. Pour résumer, tout se passe comme si une valeur de 12345,67 Euros était convertie dans le nombre entier 123456700 pour les calculs internes, puis le résultat affiché en positionnant la virgule devant les 4 derniers chiffres.

En résumé, une variable monétaire admet une précision de 4 décimales sur une gamme de valeurs entre -922 337 203 685 477,5808 et + 922 337 203 685 477,5807. Elle s'écrit sans utiliser de symbole monétaire, sans séparateur de milliers, et avec un point décimal.

```
Dim Budget As Currency
Budget = 45292790254.35
```

Basic initialise les variables `Currency` à la valeur zéro. Reprenons l'exemple des nombres réels, en employant des variables monétaires :

```
rem Code05-03.sxw   bibli : Library3
Option Explicit

Sub CalculsMonetaires()
Dim A As Currency, B As Currency, C As Currency
Dim R1 As Currency, R2 As Currency
A = -1.2345
B = 1.2346
C = -0.0001
R1 = B + A + C
R2 = B + C + A
print "R1=" & R1 ' résultat nul
print "R2=" & R2 ' résultat nul
End Sub
```

Le résultat affiché sera bien zéro pour `R1` et pour `R2`.

Avant la version 2.0 d'OpenOffice.org, seuls deux opérateurs fonctionnent pour les variables monétaires :

- `+` addition
- `-` soustraction

À partir de la version 2.0, la multiplication et la division fonctionnent, suite à la correction du rapport IZ 20704.

Néanmoins, il a été découvert une autre anomalie (rapport IZ 31001) qui n'est pas encore corrigée. Elle est mise facilement en évidence :

```
A = 200.432
print A
```

L'instruction `print` affiche : 200.4319 ! La conversion de la valeur écrite vers la valeur de type `Currency` est incorrecte. Tant que ce dysfonctionnement n'est pas corrigé, nous déconseillons d'utiliser le type `Currency`.

Ordre d'évaluation des opérateurs numériques

Les opérateurs numériques du tableau 5-3 sont présentés par ordre décroissant de priorité. Sur chaque ligne, les opérateurs ont la même priorité. L'évaluation de l'expression s'effectue de gauche à droite quand les opérateurs rencontrés sont de priorité identique. Les parenthèses permettent d'imposer un ordre d'évaluation.

Tableau 5–3 Ordre d'évaluation des opérateurs numériques

Priorité	Opérateurs
1	- (le signe moins, opérateur unaire)
2	^
3	* / mod
4	+ -

Les booléens

Les variables booléennes

Les variables du type `Boolean` ne possèdent que deux valeurs : `False` (faux) et `True` (vrai). Elles sont très utiles dans les expressions conditionnelles.

Elles sont déclarées ainsi :

```
Dim Oui As Boolean
```

Basic initialise les variables `Boolean` à la valeur `False`. Ces variables sont stockées en interne sous forme de deux octets (16 bits) alors qu'un seul bit suffirait, parce que les processeurs courants traitent plus facilement les mots de 16 bits que les éléments binaires unitaires. Dans Basic, il a été décidé que lors d'une conversion de type :

- Une valeur numérique nulle est assimilée à `False`.
- Toute autre valeur numérique est assimilée à `True`.
- `False` est converti dans la valeur numérique zéro.
- `True` est converti dans la valeur numérique -1.

Basic connait les mots `True` et `False`. Utilisez-les, votre code en sera bien plus lisible et vous éviterez ainsi de vous poser des questions sur les valeurs de vos variables booléennes.

ATTENTION

L'aide en ligne de OOo version 1.1, page « Utilisation des variables », indique : « Toute valeur assignée à une variable logique est convertie en "False" (faux) si la valeur n'est pas exactement égale à "-1" ». Ceci est incorrect.

Le code suivant montre diverses manières d'affecter des valeurs booléennes :

```
rem Code05-03.sxw   bibli : Library4 Module1
Option Explicit


Sub EvaluationBooleenne()
Dim Gagnant As Boolean
Dim Oui As Boolean, Non As Boolean

' différentes valeurs évaluées à : false
Gagnant = False
print "essai 1F : ", Gagnant
Non = False
Gagnant = Non
print "essai 2F : ", Gagnant
Gagnant = 0
print "essai 3F : ", Gagnant

' différentes valeurs évaluées à : true
Gagnant = True
print "essai 1V : ", Gagnant
Oui = True
Gagnant = Oui
print "essai 2V : ", Gagnant
Gagnant = 1
print "essai 3V : ", Gagnant
Gagnant = -1
print "essai 4V : ", Gagnant
Gagnant = 0.000000000000001
print "essai 5V : ", Gagnant
End Sub
```

Les opérateurs booléens

Les expressions booléennes utilisent des opérateurs particuliers. Dans le tableau 5-4, nous emploierons des expressions combinant deux variables ou expressions booléennes A et B.

Tableau 5–4 Les opérateurs booléens

Opérateur	Expression	Résultat
`Not`	`Not A`	(Opérateur NON) `True` si A vaut `False` `False` si A vaut `True`
`And`	`A and B`	(Opérateur ET) `True` si A et B valent `True` `False` si A vaut `False`, ou B vaut `False`, ou les deux valent `False`

Tableau 5–4 Les opérateurs booléens

Opérateur	Expression	Résultat
Or	A or B	(Opérateur OU) True si A vaut True ou si B vaut True False si A et B valent False
Xor	A Xor B	(Opérateur OU exclusif) True si A vaut True et B vaut False True si A vaut False et B vaut True False si A vaut True et B vaut True False si A vaut False et B vaut False
Eqv	A Eqv B	(Opérateur Équivalence) True si A vaut True et B vaut False True si A vaut False et B vaut False False si A vaut True et B vaut False False si A vaut False et B vaut True
Imp	A Imp B	(Opérateur Implication) False si A vaut True et B vaut False True dans les trois autres cas

L'expression `A Eqv B` est identique à l'expression `Not(A Xor B)`.
L'expression `A Imp B` est identique à l'expression `Not A Or B`. L'opérateur `Imp` est rarement utilisé.
Le code suivant visualise les différentes combinaisons de l'expression :

```
rem Code05-03.sxw   bibli : Library4 Module2
Option Explicit

Sub OperateurImp()
Dim A As Boolean, B As Boolean
Dim Resultat As Boolean

A = False
B = False
Resultat = A Imp B
print A, "Imp", B, "=", Resultat
A = False
B = True
Resultat = A Imp B
print A, "Imp", B, "=", Resultat
A = True
B = False
Resultat = A Imp B
print A, "Imp", B, "=", Resultat
A = True
B = True
Resultat = A Imp B
print A, "Imp", B, "=", Resultat
End Sub
```

Les calculs booléens sur des nombres

Ces calculs sont utilisés rarement, et pour des besoins très spécifiques. Pour comprendre cette section, le lecteur doit être familier des représentations binaires et hexadécimales.

Les opérateurs booléens `not`, `and`, `or` et `xor` peuvent être appliqués sur des variables entières. Dans ce cas, l'opération est effectuée bit à bit sur la représentation binaire du contenu des variables entières, soit sur 16 bits pour des entiers `Integer` et sur 32 bits pour des entiers `Long`.

```
rem Code05-03.sxw   bibli : Library4 Module3
Option Explicit

Sub CalculsBinaires()
Dim A As Long, B As Long

A = 513                        ' X0201
B = 776                        ' X0308
print A or B ' affiche : 777 = X0309
A = 253                        ' X00FD
B = 42                         ' X002A
print A and B ' affiche : 40 = X0028
print not A   ' affiche : -254 = XFFFFFF02
End Sub
```

Les opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer deux valeurs numériques (utilisant des entiers ou des réels) ou deux chaînes de caractères. Nous les décrivons ici parce que le résultat d'une comparaison est `True` ou `False`, c'est-à-dire un résultat booléen. La comparaison de chaînes de caractères s'effectue caractère par caractère de gauche à droite, en comparant leurs valeurs Unicode. Il en résulte que les majuscules sont distinguées des minuscules, et les caractères accentués sont différenciés. Dans le tableau 5-5, nous emploierons des expressions combinant deux variables ou expressions A et B.

Tableau 5–5 Les opérateurs de comparaison

Opérateur	Expression	Résultat
=	A = B	True si A est strictement égal à B
<>	A <> B	True si A est différent de B
<	A < B	True si A est strictement inférieur à B
<=	A <= B	True si A est inférieur ou égal à B
>	A > B	True si A est strictement supérieur à B
>=	A >= B	True si A est supérieur ou égal à B

Rappelons que la comparaison stricte entre deux variables ou expressions en nombres réels n'a pas de sens à cause des inévitables erreurs de conversions et de calculs.

Calculs booléens

Comme les opérateurs de comparaison fournissent une valeur booléenne, on peut combiner plusieurs de celles-ci avec des opérateurs booléens pour obtenir un résultat booléen.

```
rem Code05-03.sxw   bibli : Library4 Module4
Option Explicit

Sub Main4()
Dim x As Long, y As Long
Dim resultat As Boolean

x = 347
resultat = x = 347  ' style d'écriture à éviter !
print resultat
y = 6
resultat = (x = 341) or (y+x-3 >= 350)
print resultat
End Sub
```

Dans cet exemple, la première expression booléenne s'écrit plus clairement :

```
resultat = (x = 347)
```

La deuxième expression de l'exemple combine deux évaluations numériques avec l'opérateur booléen `or`. N'hésitez pas à employer des parenthèses pour forcer l'ordre d'évaluation d'une expression, et aussi pour rendre l'expression plus lisible.

Ordre d'évaluation des opérateurs booléens et de comparaison

Les opérateurs du tableau 5-6 sont présentés par ordre décroissant de priorité. Sur chaque ligne, les opérateurs ont la même priorité. L'évaluation de l'expression s'effectue de gauche à droite quand les opérateurs rencontrés sont de priorité identique.

Tableau 5–6 Ordre d'évaluation des opérateurs booléens et de comparaison

Priorité	Opérateurs
1 (maximale)	`not` (opérateur unaire)
2	`< <= = <> >= >`
3	`and or xor eqv imp`

Les parenthèses permettent d'imposer un ordre d'évaluation quelconque. N'hésitez pas à les employer.

Les variables de date

Les variables de type `Date` servent à mémoriser une date et une heure. L'implémentation du type `Date` est en fait une variable `Double` dans laquelle la partie entière représente un nombre de jours, et la partie décimale une heure précise à la seconde, exprimée en fraction de jour. Le jour du 30 décembre 1899 a la valeur zéro, le lendemain a la valeur 1. Basic initialise les variables `Date` au 30/12/1899 à 0h 0min 0s.

Le seul moyen d'initialiser une variable `Date` par programmation est d'utiliser une fonction de conversion. Il existe une fonction pour obtenir une date, une fonction pour obtenir une heure, mais aucune fonction pour initialiser simultanément date et heure. Le code ci-dessous montre un exemple simple de variables de type `Date`.

```
rem Code05-03.sxw   bibli : Library5 Module1
Option Explicit

Sub LesDates()
Dim A As Date, B As Date, C As Date

A = 15/03/2004 ' ceci ne donne pas ce que vous croyez
B = DateSerial(2004,3,15) ' date : 15/03/2004
C = TimeSerial(9,57,35)   ' heure : 09h 57mn 35s
print A, B, C
A = B + C ' 15/03/2004 à 09h 57mn 35s
print A
A = A +320 ' 320 jours plus tard, même heure
print A
End Sub
```

La première initialisation de la variable `A` est acceptée, mais pas comme la date indiquée. En effet, l'expression est évaluée mathématiquement de gauche à droite, et le résultat est interprété comme une heure puisqu'il est inférieur à 1. La deuxième affectation de valeur à la variable `A` permet d'obtenir une date et une heure. La dernière modification de la variable `A` montre comment on peut changer de jour.

La date la plus ancienne qu'on puisse assigner avec `DateSerial` est l'année 100, bien que cela n'ait pas grand sens puisque le calendrier Grégorien débute en 1582. La date la plus futuriste acceptée est le 31/12/9999. Notez que dans les boîtes de dialogue, le contrôle de champ Date se limite à la période du 1er janvier 1600 au 1er janvier 9999.

Les dates peuvent être comparées. On peut ajouter ou retrancher un nombre de jours ou une valeur d'heure.

Il existe diverses fonctions pour obtenir ou afficher des dates et heures, nous les verrons dans le chapitre 8.

Les objets

Les variables de type `Object` sont utilisées pour accéder à des entités qui ne correspondent à aucun des types connus de Basic. Nous utiliserons ce type de variables à partir du chapitre 10, lorsque nous utiliserons l'API OpenOffice.org. En voici un exemple typique :

```
Dim monDocument As Object
monDocument = ThisComponent
```

Il existe une forme spéciale de déclaration d'objet, qui sert à obtenir une variable conforme à un « modèle ».

```
Dim Prop As New com.sun.star.beans.PropertyValue
```

Le terme `New` demande à Basic de créer une nouvelle variable selon le modèle qui suit. Ce modèle est désigné par un nom composé, qui doit être écrit en respectant les majuscules et minuscules. Nous en apprendrons plus sur les noms composés lorsque nous aborderons l'API.

Le type Variant

Les variables de type `Variant` sont les plus souples, car elles peuvent contenir des données des différents types déjà vus. Une telle variable peut changer de type au fil des instructions.

> **DOC Aide en ligne de Basic**
>
> L'aide en ligne traduit `Variant` par « variante » dans le corps du texte. Nous préférons garder le terme anglais, avec la majuscule.

Pourquoi utiliser des `Variant` ? Si vous concevez une routine dont un argument est déclaré de type `Variant`, votre routine peut être appelée avec un argument aussi bien de type `String` que de type `Double` ou `Long`, etc. De même, si vous définissez une fonction comme renvoyant un type `Variant`, tout type de résultat est possible en fonction de l'exécution. Enfin, nous le verrons dans d'autres chapitres, le type `Variant` est parfois indispensable avec certaines interfaces de l'API pour lesquelles on ne peut connaître le type de la variable qu'en cours d'exécution.

Basic admet trois formes pour déclarer une variable de type `Variant` :

```
                          ' on ne déclare pas la variable ! !
Dim maVariable            ' déclaration implicite du type
Dim maVariable As Variant ' déclaration explicite du type
```

Les deux premières formes ont déjà été signalées quand nous avons expliqué les déclarations de variables. Elles sont à éviter si on souhaite écrire un code clair.

La valeur Empty

Basic initialise les variables `Variant` à une pseudo-valeur : `Empty`. Ceci indique que la variable ne contient pas de donnée, mais qu'elle est disponible. Notez que la valeur `Empty` n'est pas disponible en programmation. On contourne le problème avec une autre variable non initialisée :

```
rem Code05-03.sxw   bibli : Library6 Module1
Option Explicit

Sub Main1()
Dim aa As Variant, vide As Variant

aa = 3
print "Valeur de aa : ", aa
aa = vide
print "Valeur de aa : ", aa ' ceci n'affiche rien
if aa = "" then print "Chaîne vide"
if aa <> "" then print "Chaîne non vide"
if aa = 0 then print "Alors aa est nul ?"
if IsEmpty(aa) then print "aa est vide"
End Sub
```

L'exécution de cet exemple montre plusieurs points remarquables :

- Le premier `print` montre bien une valeur numérique.
- Une fois la valeur `Empty` affectée à la variable `aa`, l'instruction `print` n'affiche plus rien, comme pour une chaîne vide (c'est-à-dire de longueur nulle).
- L'instruction suivante n'affiche rien, donc `aa` n'est pas considérée comme une chaîne vide.
- L'instruction suivante n'affiche rien non plus, donc `aa` n'est pas non plus une chaîne non vide !
- Le test `aa = 0` donne `True`. Ceci provient d'une conversion implicite de la valeur `Empty` en une valeur équivalente pour un nombre entier, qui est zéro.
- Finalement, la fonction `IsEmpty()` renvoie `True`, ce qui confirme que `aa` contient la valeur `Empty`.

La valeur Null

Le type `Variant` dispose d'une autre pseudo-valeur : `Null`. Elle indique que la variable ne contient pas de donnée, ce qui est différent de la valeur `Empty`. La valeur `Null` peut être employée pour renvoyer une donnée signifiant « pas d'information » :

```
Dim resu As Variant
' fin d'un traitement
resu = Null
```

Vous ne pouvez pas utiliser une variable ayant la valeur `Null`. Les instructions suivantes provoquent des erreurs :

```
Dim aa As Variant, bb As Long
aa = Null
bb = aa  ' erreur d'exécution car bb n'est pas un Variant
print aa ' erreur d'exécution
if IsNull(bb) then print "aa est Null" ' fonctionne
```

La dernière instruction utilise `IsNull()` qui est une des fonctions Basic que nous allons maintenant décrire.

Comment connaître le type d'un Variant ?

Sachant qu'une variable est de type `Variant`, comment savoir ce qu'elle représente à un instant d'exécution du programme ? Basic fournit à cet effet plusieurs fonctions d'interrogation, listées dans le tableau 5-7. Toutes ces fonctions renvoient un résultat `Boolean` : soit `True` (vrai), soit `False` (faux).

Tableau 5–7 Fonctions d'interrogation de type

Fonction	Signification quand le résultat est : True (vrai)
`IsNull()`	la variable ne contient pas de donnée
`IsEmpty()`	la variable est vide
`IsNumeric()`	la variable est un nombre (entier ou réel)
`IsDate()`	la variable est une date ou une heure
`IsArray()`	la variable est un tableau
`IsObject()`	la variable est un objet

Voici un exemple utilisant ces fonctions :

```
rem Code05-03.sxw   bibli : Library6 Module2
Option Explicit
```

```
Sub Main2()
Dim toto As Variant
Dim resu As Boolean
if IsEmpty(toto) then print "vide" else print "pas vide !"
toto = Null
if IsNull(toto) then print "nul" else print "pas nul !"
toto = 3.14
resu = IsNumeric(toto)
print "Nombre ? " & resu
resu = IsDate(toto)
print "Date ? " & resu
End Sub
```

Ces fonctions n'ont de sens qu'avec des variables `Variant`. Démonstration :

```
rem Code05-03.sxw   bibli : Library6 Module3
Option Explicit
Sub Main3()
Dim aa as Long
Dim resu as boolean
resu = IsEmpty(aa)
print resu
End Sub
```

La routine affiche : `False`.

OpenOffice.org Basic dispose de deux fonctions permettant de connaître le type d'une variable :

- `TypeName()` renvoie une chaîne de caractères précisant le type.
- `VarType()` renvoie une valeur numérique précisant le type.

Le code suivant donne quelques exemples.

```
rem Code05-03.sxw   bibli : Library6 Module4
Option Explicit

Sub Main4()
Dim aa As Variant, bb As Long, cc As String

print VarType(aa), TypeName(aa) ' affiche : 0 empty
aa = 12345
print VarType(aa), TypeName(aa) ' affiche : 2 Integer
print VarType(bb), TypeName(bb) ' affiche : 3 Long
print VarType(cc), TypeName(cc) ' affiche : 8 String
End Sub
```

Le tableau 5-8 liste les valeurs possibles.

Tableau 5–8 Valeurs renvoyées par VarType et TypeName

VarType	TypeName	Type de donnée
8	String	Chaîne de caractères
2	Integer	Entier
3	Long	Entier
4	Single	Réel
5	Double	Réel
6	Currency	Monétaire
11	Boolean	Booléen
7	Date	Date
9	Object	Objet
12	Variant	Variant
0	Empty	Variant, valeur à la déclaration
1	Null	Valeur non disponible

Les constantes

Les constantes servent à nommer une valeur fixe, connue à l'écriture du programme.

Une constante s'écrit avec le mot-clé `Const` suivi d'une définition ou de plusieurs séparées par des virgules. Chaque définition s'écrit comme si on initialisait une variable avec une valeur littérale.

```
rem Code05-03.sxw   bibli : Library7 Module1
Option Explicit

Sub Main1()
Dim a As Long, b As Double
Dim p As String
Const a2 = -314789, poete = " Charles Baudelaire"
Const e = 2.718281828459045

a = a2
b = Pi
print a, b
b = e
p = poete
print e, p
End Sub
```

Nous avons utilisé un cas particulier : `Pi` est une *constante prédéfinie* dans Basic.

Basic affiche un message d'erreur si on essaie de modifier la valeur d'une constante. L'usage de constantes rend le code plus robuste que si on écrivait des valeurs littérales au fil des instructions.

Une constante `Private`, ou `Public`, ou `Global` s'écrit en précédant le mot `Const` d'un de ces termes et en plaçant la ligne avant les routines.

```
Private Const a2 = -314789, poete = "Ronsard"
Public Const a3 = "Bonjour"
Global Const a4 = 5.7e6
```

BOGUE Constantes partagées

Si vous définissez une constante en dehors d'une routine, n'oubliez pas d'ajouter un des termes `Private`, `Public` ou `Global`. Sinon vous pourrez rencontrer des comportements anormaux.

Les tableaux

Jusqu'à maintenant, nous avons étudié des variables simples, qui ont une seule valeur à un moment donné. Les variables tableaux (en anglais : Array) représentent non pas une mais plusieurs valeurs simultanées. Elles doivent être déclarées explicitement. Les variables tableaux existent dans tous les types de données déjà rencontrés.

L'aide en ligne de Basic et la documentation Sun Microsystems traduisent le terme anglais *Array* par le mot français Matrice, mais nous préférons et utiliserons le mot de *Tableau*.

Les tableaux unidimensionnels (vecteurs)

Il s'agit du tableau le plus simple : un ensemble de valeurs accessibles avec un seul index. Il existe plusieurs manières de déclarer un tableau. Voici la plus courante à travers un exemple :

```
rem Code05-04.sxw   bibli : Library1 Module1
Option Explicit

Sub Main1()
Dim Temperatures(23) As Double
```

```
Temperatures(0) = 4.5
Temperatures(15) = 13.2
Temperatures(23) = 6
End Sub
```

L'instruction `Dim` déclare une variable `Temperatures` qui est destinée à contenir la mesure de température de la journée, heure par heure. Cette déclaration réserve 24 zones mémoire de type `Double`. On accède à chaque zone avec un index entre 0 et 23, comme indiqué dans les lignes suivantes du code.

Dans cette méthode de déclaration, le nombre entre parenthèses est la valeur la plus élevée utilisable de l'index. La valeur la plus basse est zéro, ce qui nous donne ici 24 zones.

La deuxième méthode nous permet de fixer des valeurs arbitraires aux valeurs extrêmes de l'index. Nous allons supposer que nous mémorisons la température moyenne de chaque mois de l'année :

```
rem Code05-04.sxw   bibli : Library1 Module2
Option Explicit

Sub Main2()
Dim Temperatures(1 to 12) As Double

Temperatures(1) = 5
Temperatures(5) = 13
Temperatures(8) = 27
Temperatures(12) = 6
End Sub
```

La déclaration de tableau précise les valeurs extrêmes de l'index. Il est possible d'utiliser des valeurs négatives. La valeur après le `to` doit être supérieure ou égale à la valeur avant le `to`.

> **REMARQUE** **Taille des tableaux**
>
> Les valeurs extrêmes de l'index d'un tableau peuvent être quelconques. Le nombre maximal d'éléments d'un tableau ne dépend que de la mémoire disponible. Toutefois, pour des raisons d'implémentation, chaque élément occupe au moins 100 octets en mémoire. Ainsi, un tableau d'un million d'éléments `Boolean` occupera environ 100 Mega-octets !

Les tableaux multidimensionnels

Supposons que nous ayons besoin de mémoriser la température de chaque jour de l'année. Nous avons deux solutions : soit utiliser un vecteur avec 366 éléments (pour tenir compte des années bissextiles), soit utiliser un tableau à deux dimensions :

```
rem Code05-04.sxw   bibli : Library1 Module3
Option Explicit

Sub Main3()
Dim Temperatures(1 to 12, 1 to 31) As Double

Temperatures(1, 1) = 5   ' 1er janvier
Temperatures(5, 30) = 13 ' 30 mai
Temperatures(8, 15) = 27 ' 15 août
Temperatures(12, 31) = 6 ' 31 décembre
End Sub
```

Dans la déclaration, nous avons indiqué deux index : le premier pour le numéro du mois, le deuxième pour le numéro du jour dans le mois. Le contenu de la variable `Temperatures` pourrait être présenté sous la forme d'un tableau de 12 lignes et 31 colonnes, avec une valeur dans chaque case. Enfin, pas tout à fait... car par exemple le 30 février n'existe pas ! Ces cases resteront inutilisées.

La première forme de déclaration peut aussi être utilisée dans la déclaration d'un tableau multidimensionnel, pour un ou plusieurs des index. Voici comment mémoriser la température de chaque heure pour chaque jour de l'année :

```
rem Code05-04.sxw   bibli : Library1 Module4
Option Explicit

Sub Main4()
Dim Temperatures(1 to 12, 1 to 31, 23) As Double

Temperatures(1, 1, 0) = 5   ' 1er janvier à 00h
Temperatures(8, 15, 14) = 27 ' 15 août à 14h
End Sub
```

On dit que le tableau a trois dimensions. Basic permet un nombre de dimensions presque quelconque (dans la mesure des capacités mémoire), mais dans la pratique, il est rare d'en dépasser 4.

Les tableaux peuvent être définis en `Private`, `Public`, `Global`, comme les variables simples.

Redimensionner un tableau

Il y a des situations où le programmeur ne peut pas savoir au moment du codage quelle doit être la taille nécessaire pour le tableau. Il n'est pas toujours raisonnable de réserver pour toute la durée du programme un tableau avec la taille maximale dont on pourrait avoir besoin.

Basic permet de redimensionner un tableau en cours d'exécution du programme, à condition que le tableau soit *interne à une routine*. Pour cela, on utilise l'instruction `Redim`.

Cet exemple utilise un vecteur, mais un tableau multidimensionnel aurait convenu.

```
rem Code05-04.sxw   bibli : Library2 Module1
Option Explicit

Sub Main1()
Dim maTable(3) As Long, Xmax As Long

maTable(1) = 100
maTable(3) = 333
print maTable(1), maTable(2), maTable(3)

' quelques instructions plus tard...
Xmax = 9
Redim maTable(Xmax) As Long
maTable(7) = 7777
print maTable(1), maTable(2), maTable(3), maTable(7)
End Sub
```

Dans ce code, `maTable` est déclarée comme vecteur d'index 0 à 3. La première instruction `print` affiche 100, 0, 333 car l'élément d'index 2 a gardé sa valeur d'initialisation. L'instruction `Redim` transforme `maTable` en vecteur d'index 0 à `Xmax`, qui vaut 9. Attention, Basic refuse de changer le type des éléments du vecteur, ou le nombre de dimensions du tableau.

Vous remarquerez que la deuxième instruction `print` affiche des zéros pour les anciennes valeurs : le redimensionnement a éliminé les valeurs qui préexistaient. Heureusement, nous pouvons éviter ceci, en utilisant l'option `Preserve`.

```
rem Code05-04.sxw   bibli : Library2 Module2
Option Explicit

Sub Main2()
Dim maTable(3) As Long, Xmax As Long

maTable(1) = 100
maTable(3) = 333
print maTable(1), maTable(2), maTable(3)
' quelques instructions plus tard...
Xmax = 9
Redim Preserve maTable(Xmax) As Long
maTable(7) = 7777
print maTable(1), maTable(2), maTable(3), maTable(7)
End Sub
```

Maintenant, les valeurs préexistantes sont gardées malgré le redimensionnement. Évidemment, il faut pour cela que l'ancien dimensionnement et le nouveau soient compatibles en terme d'index.

> **ATTENTION** **Redimensionnement et vitesse d'exécution**
>
> Le redimensionnement d'un tableau représente un certain travail pour Basic, en particulier si on utilise `Preserve`. De nombreux redimensionnements peuvent sérieusement ralentir votre programme.

Connaître les limites d'index

Dans certaines situations, on se retrouve avec une variable tableau dont on ne connaît pas la plage des index valides. Basic nous fournit deux fonctions pour nous tirer d'affaire :

- `LBound()` renvoie la valeur minimale de l'index.
- `UBound()` renvoie la valeur maximale de l'index.

L'exemple suivant est assez artificiel, mais montre comment utiliser ces fonctions.

```
rem Code05-04.sxw   bibli : Library2 Module3
Option Explicit

Sub Main3()
Dim IndexMin As Long, IndexMax As Long
Dim unVecteur(75 to 139) As Double
Dim uneTable(5, -3 to 7, 15) As Boolean

IndexMin = LBound(unVecteur())
IndexMax = UBound(unVecteur())
print "Vecteur", IndexMin, IndexMax

IndexMin = LBound(uneTable())
IndexMax = UBound(uneTable())
print "Table", IndexMin, IndexMax ' premier index

IndexMin = LBound(uneTable(), 1)
IndexMax = UBound(uneTable(), 1)
print "Table1", IndexMin, IndexMax ' premier index

IndexMin = LBound(uneTable(), 2)
IndexMax = UBound(uneTable(), 2)
print "Table2", IndexMin, IndexMax ' deuxième index

IndexMin = LBound(uneTable(), 3)
IndexMax = UBound(uneTable(), 3)
print "Table3", IndexMin, IndexMax ' troisième index
End Sub
```

Dans l'utilisation de `LBound()` et `UBound()`, il est nécessaire de fournir en argument la totalité du tableau. Pour que Basic comprenne notre intention, il est nécessaire d'écrire le nom du tableau suivi de deux parenthèses sans rien à l'intérieur.

Le deuxième argument, facultatif, de ces fonctions est le numéro de la dimension, ou rang de l'index dont on recherche la borne. Pour un vecteur, ce rang est égal à 1. Dans le

cas d'un tableau multidimensionnel, il vaut entre 1 et n, le nombre de dimensions du tableau. Si vous dépassez le nombre de dimensions, une erreur sera déclenchée. Cela peut être un moyen de connaître le nombre de dimensions, si vous savez gérer les erreurs.

Les affectations entre tableaux

Cette particularité de Basic a été signalée par Danny Brewer (*DannyB*) dans le forum de langue anglaise OOoForum http://www.oooforum.org/. Le point de départ est la faculté d'affecter un tableau à un autre tableau de mêmes dimensions, en utilisant la syntaxe vue dans la section précédente :

```
S4() = S3()
```

La macro ci-dessous doit être exécutée pour afficher les messages.

```
rem Code05-04.sxw   bibli : Library2 Module4
Option Explicit

Sub PointerTableau()
Dim S1 As String, S2(1) As String
Dim S3(2) As String, S4(2) As String

S3(0) = "un"
S3(1) = "deux"
S3(2) = "trois"

S4() = S3() ' l'intention est de copier le tableau S3()
S1 = S3(1) ' copie d'un élément de S3()
' copie d'un élément de S3() dans un élément de S2()
S2(1) = S3(1)
print "Avant", S1, S2(1), S3(1), S4(1)

S3(1) = "modifié" ' changeons un élément de S3()
print "Après", S1, S2(1), S3(1), S4(1)
' conclusion : S3 et S4 pointent sur le même tableau
' par contre S1 et S2(1) sont bien des copies
S4(1) = "changé" ' confirmation en modifiant S4
print "Après", S1, S2(1), S3(1), S4(1)
End Sub
```

Pour un débutant en programmation, le résultat tient de la magie. Le programmeur expérimenté reconnaît un effet de pointeur. Dans l'implémentation, la zone mémoire du tableau se trouve à une certaine adresse, et ce que nous nommons `S3` et `S4` ne contiennent que l'adresse mémoire du tableau.

L'avantage est qu'il n'y a pas de recopie, simplement la mise à jour du pointeur. L'inconvénient est que si nous voulons réellement dupliquer le tableau, il faut s'y prendre autre-

ment. Notre exemple nous a mis sur la voie : il faut recopier chaque élément du tableau, un par un. Cette opération est réalisée par le code suivant, qui utilise l'instruction de boucle `for` que nous verrons au chapitre 6.

```
rem Code05-04.sxw   bibli : Library2 Module5
Option Explicit

Sub CopierTableau()
Dim S3(2) As String, S4(2) As String
Dim x As Long

S3(0) = "un"
S3(1) = "deux"
S3(2) = "trois"
' recopie du tableau élément par élément
for x = LBound(S3()) to UBound(S3())
  S4(x) = S3(x)
next
S3(1) = "modifié"
print S3(1), S4(1) ' Vérification
End Sub
```

Les Variant et les tableaux

Les variables `Variant` ont des propriétés tout à fait particulières dans le domaine des tableaux. Ces propriétés les rendent indispensables dans certains échanges d'information avec l'API d'OpenOffice.org.

Les tableaux de Variant

Un tableau de `Variant` peut être déclaré comme le serait un autre tableau :

```
Dim Table1(5,3) As Long
Dim Table2(5,3) As Variant
```

Cependant, il existe une grande différence pour le programmeur : la variable `Table1` ne peut contenir que des valeurs de type `Long`, alors que la variable `Table2` peut contenir simultanément des éléments de type différents. Nous pouvons écrire par exemple :

```
Table2(3,1) = -576
Table2(0,0) = "Mercredi"
Table2(0,1) = 6.55957
```

Et comme pour une variable `Variant` simple, nous pourrions changer le type d'un élément au cours du programme.

La fonction Array()

Cette fonction est très pratique pour remplir en une seule instruction une variable vecteur avec des valeurs. Si vous êtes très attentif, cette macro devrait vous surprendre :

```
rem Code05-04.sxw   bibli : Library2 Module6
Option Explicit

Sub VecteurBizarre()
Dim unVecteur As Variant
Dim y As Double

y = 12345678.9
unVecteur = array(55555, "Hello", y)
print unVecteur(0), unVecteur(1), unVecteur(2)
End Sub
```

D'une part, nous avons défini `unVecteur` comme une variable simple. Et pourtant, nous obtenons grâce à la fonction `Array()` un vecteur à trois éléments ! Dans la déclaration, nous n'avons pas eu besoin de préciser la taille de l'index. C'est d'un grand intérêt si vous devez récupérer d'une fonction de l'API un tableau dont vous ne savez rien.

D'autre part, nous avons créé un vecteur contenant un élément entier, un élément chaîne de caractères et un élément nombre réel. Ceci est possible car chaque élément est un `Variant`.

Les tableaux irréguliers

Comme `Array()` accepte des valeurs calculées à l'exécution, nous pouvons essayer de lui donner en argument un vecteur ! Voilà ce que cela donne :

```
rem Code05-04.sxw   bibli : Library2 Module7
Option Explicit

Sub TableauIrregulier()
Dim v1 As Variant, v2 As Variant, v3 As Variant
Dim m1 As Variant, extrait As Variant
v1 = array(12345, 11)
v2 = array(43210, 22, 777, 999)
v3 = array(55555, 3.14, "Hello")
m1 = array(v1,v2,v3) ' construction du tableau
extrait = m1(1)

' cette instruction affiche : 3 43210 999
print UBound(extrait), extrait(0), extrait(3)
extrait = m1(2)

' cette instruction affiche : 2 55555 Hello
print UBound(extrait), extrait(0), extrait(2)
End Sub
```

Nous avons obtenu un étrange tableau composé d'un vecteur à deux éléments, un vecteur à quatre éléments et un vecteur à trois éléments. Il n'est pas possible d'obtenir directement un élément du tableau `m1` : l'instruction `m1(1,1)` déclencherait une erreur. En revanche, en utilisant les propriétés des `Variant`, on peut récupérer un sous-ensemble, et ainsi de suite jusqu'à obtenir un vecteur. Chaque élément du vecteur peut alors être récupéré.

Diverses fonctionnalités concernant les variables Basic

Basic possède quelques instructions destinées à simplifier l'écriture de programmes. Nous les signalons afin que vous puissiez les reconnaître dans vos lectures, mais nous déconseillons leur usage. Le gain apparent à l'écriture sera annulé par une perte de temps bien plus grande à la mise au point.

Les caractères de déclaration de type

Dans une déclaration `Dim`, il est possible de déclarer une variable d'un type donné sans l'écrire explicitement. Pour cela, le nom de la variable doit être terminé par un caractère spécial. Ces caractères sont listés dans le tableau 5-9.

Tableau 5–9 Liste des caractères de type

Caractère	Exemple	Type de donnée
$	`Dim unNom$`	`String`
%	`Dim age%`	`Integer`
&	`Dim population&`	`Long`
!	`Dim distance!`	`Single`
#	`Dim dimension#`	`Double`
@	`Dim budget@`	`Currency`
aucun	-	`Boolean`
aucun	-	`Date`
aucun	-	`Object`
aucun	-	`Variant`

Les caractères de type pour variables non déclarées

Quand on a la mauvaise habitude de ne pas déclarer les variables, il faut un autre moyen pour indiquer le type de la variable. Basic utilise dans ce but la première lettre du nom de

la variable. Cette lettre peut être définie avec une instruction de la forme `Defxxx`. Par exemple, ceci définit les lettres v et z comme lettres désignant le type `Variant` :

```
DefVar v, z
```

L'instruction `Defxxx` doit être ajoutée avant le code du module. Par défaut, aucune lettre n'est définie, donc toute variable est du type `Variant`. Malgré cela, certains programmeurs ont l'habitude d'utiliser certaines lettres pour indiquer le type de la variable. Ils mettent cette lettre en minuscule, suivie d'une majuscule pour le deuxième caractère du nom de la variable.

Le tableau 5-10 liste la lettre usuelle et l'instruction pour chaque type. Cette manière de nommer les variables est assez répandue dans les exemples de code utilisant les objets de l'API :

```
bSuccess = HasUnoInterfaces(oSimpleFileAccess,IfaceName3$)
```

Tableau 5–10 Lettre usuelle et instruction de définition de type

Lettre usuelle	Instruction	Type de donnée
s	DefStr	String
i	DefInt	Integer
l	DefLng	Long
aucun	DefSng	Single
d	DefDbL	Double
aucun	DefCur	Currency
b	DefBool	Boolean
t	DefDate	Date
o	DefObj	Object
v	DefVar	Variant

Pour autant, nous insistons sur l'intérêt de déclarer préalablement toute variable que vous utilisez, en précisant explicitement le type. Dans ce cas, ce caractère est inutile ainsi que celui décrit dans la section précédente.

DimArray()

Quand on a la mauvaise habitude de ne pas déclarer les variables, il faut un autre moyen pour définir un tableau. Pour cela, Basic dispose de la fonction `DimArray` qui renvoie un tableau de `Variant`.

```
rem Code05-04.sxw   bibli : Library2 Module8
```

```
Sub Main8()

unTableau = DimArray(8,6,11, 2)
print UBound(unTableau(), 1), UBound(unTableau(), 2),_
      UBound(unTableau(), 3), UBound(unTableau(), 4)
End Sub
```

Les arguments de `DimArray` ont la même signification que pour une déclaration de tableau avec `Dim`. On ne peut pas préciser explicitement la borne inférieure de chaque index.

Option Base

Si elle est utilisée, l'instruction `Option Base` doit être ajoutée avant le code du module. Elle sert à définir la limite inférieure par défaut des index de tableau. Vous avez le choix entre 0 et 1.

```
Option Base 1
```

L'instruction ci-dessus définit la limite inférieure à 1, alors que par défaut la limite est 0. La valeur par défaut est équivalente à écrire :

```
Option Base 0
```

Il est déconseillé de changer la valeur par défaut, car la plupart des programmeurs Basic y sont habitués, le gain éventuel en mémoire est totalement négligeable et enfin l'API utilise des index à base zéro.

Conclusion

OOoBasic sait manipuler des entités et des variables de natures différentes. Chacunes ont leurs spécificités propres : numériques ou chaînes de caractères, elles ont toutes un domaine de définition et d'utilisation identifié.

Les variables peuvent être manipulées sous forme de tableaux dont nous avons exposé les principes et la manipulation.

Le chapitre suivant nous permettra d'utiliser ces éléments à travers des instruction conditionnelles et itératives.

6

Conditions, boucles et branchements

Les instructions Basic que nous allons décrire ici permettent de modifier le comportement d'un programme en lui faisant choisir une série d'instructions plutôt qu'une autre, suivant certaines conditions.

Les têtes de section de ce chapitre reprennent les noms Basic de ces instructions.

If Then Else

C'est l'instruction conditionnelle la plus courante. Les mots anglais `If`, `Then`, `Else` veulent dire respectivement Si, Alors, Sinon. La signification de cette instruction correspond à une phrase équivalente en langage courant.

```
If expr1 Then
  instructions1v
Else
  instructions1f
End If
```

Le mot `Then` doit être sur la même ligne que le `If` (sauf si on utilise le caractère de continuation de ligne). Les mots `Else` et `End If` doivent être écrits sur des lignes différentes, et seuls sur la ligne. On peut écrire `EndIf` au lieu de `End If`.

Si `expr1` est vraie, la ou les `instructions1v` est(sont) exécutée(s) ; si `expr1` est fausse, la ou les `instructions1f` est(sont) exécutée(s). `Expr1` peut être aussi bien une simple variable booléenne, qu'une expression complexe donnant un résultat booléen.

Il doit y avoir au moins une instruction après le `Then`. La deuxième partie : `Else`, `instructions1f` peut être omise, mais l'instruction doit toujours se terminer par `End If`.

```
If expr1 Then
  instructions1v
End If
```

Lorsqu'une seule instruction suffit dans la partie `Then` et dans la partie `Else` la séquence peut s'écrire sur une seule ligne :

```
If expr1 Then instruction1v Else instruction1f
```

Et si la partie `Else` n'est pas utilisée :

```
If expr1 Then instruction1v
```

Notez l'absence du `End If` dans ces deux formes simplifiées.

Une instruction `If` peut être imbriquée à l'intérieur d'une autre instruction `If`, dans la première ou la deuxième partie.

```
If expr1 Then
  If expr2 Then
    instructions2v
  Else
    instructions2f
  EndIf
Else
  instructions1f
End If
```

`Expr1` et `expr2` sont des expressions booléennes quelconques. Notez que `expr2` n'est pas évaluée si `expr1` est fausse. Ceci peut avoir de l'importance dans certaines expressions.

L'imbrication peut se répéter à l'image de poupées russes. En revanche, les séquences ne doivent pas se chevaucher ; c'est pourquoi il est fortement conseillé d'écrire ces instructions avec une indentation (un décalage d'espaces) pour visualiser les niveaux d'imbrication. L'imbrication dans la deuxième partie du `If` donne :

```
If expr1 Then
  instructions1v
Else
```

```
  If expr2 Then
    instructions2v
  Else
    instructions2f
  End If
End If
```

Ceci étant assez courant, Basic fournit la variante `ElseIf` (en un seul mot) qui allège l'écriture :

```
If expr1 Then
  instructions1v
ElseIf Expr2 Then
  instructions2v
Else
  instructions2f
End If
```

L'intérêt est de pouvoir mettre en cascade des `ElseIf` pour tester plusieurs cas à la suite :

```
If expr1 Then
  instructions1v
ElseIf expr2 Then
  instructions2v
ElseIf expr3 Then
  instructions3v
Else
  instructions3f
End If
```

Dans ce dernier exemple :

- Les `instructions1v` seront exécutées si `expr1` est vraie.
- Les `instructions2v` seront exécutées si `expr1` est fausse et `expr2` est vraie.
- Les `instructions3v` seront exécutées si `expr1` et `expr2` sont fausses et `expr3` vraie.
- Les `instructions3f` seront exécutées si `expr1`, `expr2` et `expr3` sont simultanément fausses.

Dans les cas où les conditions consistent à comparer une même variable avec diverses valeurs, l'instruction `Select Case`, que nous verrons plus loin, est plus appropriée.

Voici un petit exemple pour illustrer des tests `If` imbriqués.

```
rem Code06-01.sxw   bibli : Conditions Module1
Option Explicit

Sub Main1()
Dim Homme As Boolean, NombreSpectateurs As Long
Dim Salut As String
```

```
' essayez diverses valeurs
Homme = False
NombreSpectateurs = 1

If NombreSpectateurs > 0 Then
  If Homme Then
    If NombreSpectateurs < 2 Then
      Salut = "Monsieur"
    Else
      Salut = "Messieurs"
    End If
  Else
    If NombreSpectateurs = 1 Then
      Salut = "Madame"
    Else
      Salut = "Mesdames"
    End If
  End If
  print "Bonjour " & Salut
End If
End Sub
```

Select Case

L'instrution `Select Case` signifie : Sélectionner le Cas. Le pseudo-codage qui suit liste toutes les variantes possibles :

```
Select Case expr0
Case expr1
  instructions1
Case expr2a, expr2b
  instructions2
Case expr3a to expr3b
  instructions3
Case Is OpérateurDeComparaison expr4
  instructions4
' autres instructions Case
' éventuellement
Case Else
  instructionsN
End Select
```

Les mots-clés doivent être sur des lignes différentes, comme indiqué. Il doit y avoir au moins un `Case expr`, alors que la partie `Case Else` est optionnelle. Les termes `expr0`, `expr1`, etc. représentent des expressions qui fournissent une valeur booléenne ou numé-

rique, ou une chaîne de caractères. Chaque `Case` est suivi par une ou plusieurs instruction(s).

L'instruction `Select Case` se déroule ainsi :

1. La valeur de `expr0` est comparée à la valeur de `expr1`. S'il y a égalité, les instructions `instructions1` sont exécutées, puis l'exécution continue après le `End Select`.
2. S'il n'y a pas eu égalité, le `Case` suivant est testé. La valeur de `expr0` est comparée à la valeur de chacune des `expr2` (il peut y en avoir un nombre quelconque). Si `expr0` est égal à une des `expr2` alors les instructions `instructions2` sont exécutées, puis l'exécution continue après le `End Select`.
3. S'il n'y a pas eu égalité, le `Case` suivant est testé. Si la valeur de `expr0` est comprise entre les valeurs `expr3a` et `expr3b`, les instructions `instructions3` sont exécutées, puis l'exécution continue après le `End Select`.
4. S'il n'y a pas eu égalité, le `Case` suivant est testé. Le `Is` (anglais : « Est ») représente `expr0`, qui est comparée à `expr4`. Si le résultat de la comparaison est vrai, les instructions `instructions4` sont exécutées, puis l'exécution continue après le `End Select`.
5. Le processus se poursuit sur les `Case` suivants tant qu'il n'y a pas d'égalité.
6. Le `Case Else`, qui est placé en dernier, est déclenché en dernier ressort. Il n'est pas obligatoire.

Un exemple va montrer les différentes possibilités. Changez les valeurs de `x` et `y` pour déclencher les différentes instructions.

```
rem Code06-01.sxw   bibli : Conditions Module2
Option Explicit

Sub Main2()
Dim x As Long, y As Long

y = 10
x = 1
Select Case x+7 ' expression donnant une valeur
Case 3          ' valeur simple
  Print "Case 3"
Case 7,9        ' liste de valeurs
  Print "Case 7,9"
Case 8 to 15    ' gamme de valeurs
  Print "Case 8 to 15"
Case y+12       ' expression donnant une valeur
  Print "Case y+12"
Case Is > y+12 ' comparaison d'expressions
  Print "Case Is > y+12"
Case Else
  Print "autre valeur"
End Select
End Sub
```

Le `Case 7,9` se déclenche sur l'égalité avec une des valeurs de la liste.

Le `Case 8 To 15` se déclenche sur l'ensemble des valeurs de la gamme. Notez que si `x` était une variable `Double`, une valeur comme `x = 6.35` déclencherait le `Case`.

Le `Case Is` est équivalent, localement, à :

```
If (x+7) > (y+12) Then Print "Case Is > y+12"
```

Bien que le `Is` puisse être omis, il est plus clair de le mettre quand on utilise cette forme.

Un autre exemple aurait pu utiliser des expressions chaînes de caractères au lieu d'expressions numériques.

IIf

Ce nom bizarre est probablement l'acronyme de « Immediate If ». L'instruction `IIf` peut alléger la programmation par rapport à l'usage d'un `If` classique. La syntaxe est :

```
Resultat = IIf(expr0, expr1, expr2)
```

Si `expr0` est vraie, `Resultat` reçoit la valeur résultant de `expr1`.

Si `expr0` est fausse, `Resultat` reçoit la valeur résultant de `expr2`. L'instruction est donc équivalente à :

```
If expr0 Then Resultat = expr1 Else Resultat = expr2
```

Un exemple simple :

```
rem Code06-01.sxw   bibli : Conditions Module3
Option Explicit

Sub Main3()
Dim NombreSpectateurs As Long
Dim Salut As String

NombreSpectateurs = 1
  Salut = IIf( NombreSpectateurs = 1, "Monsieur", "Messieurs")
  print "Bonjour " & Salut
End Sub
```

Ici, les expressions sont très simples, mais vous pouvez en choisir de bien plus complexes, comportant même des `IIf`. C'est ce que nous allons faire en reprenant complètement l'exemple du `If Then Else` sous une autre forme :

```
rem Code06-01.sxw   bibli : Conditions Module4
Option Explicit

Sub Main4()
Dim Homme As Boolean, NombreSpectateurs As Long
Dim Salut As String

' essayez diverses valeurs
Homme = False
NombreSpectateurs = 1

If NombreSpectateurs > 0 Then
  Salut = IIf( NombreSpectateurs = 1, _
               IIf(Homme, "Monsieur", "Madame"), _
               IIf(Homme, "Messieurs", "Mesdames") )
  print "Bonjour " & Salut
End If
End Sub
```

Pour des raisons de mise en page, la ligne du `IIf` est répartie sur plusieurs lignes avec le caractère de continuation. L'indentation a été choisie pour mettre en évidence les différentes parties de l'instruction qui affectent une valeur à la variable `Salut`. Évitez cependant de trop compliquer vos expressions, cela vous épargnera de longues recherches d'erreurs.

Vous pouvez tout à fait utiliser dans un `IIf` des `expr2` et `expr3` donnant des valeurs de types différents, si vous y trouvez un usage judicieux. L'exemple suivant est quant à lui un peu artificiel.

```
rem Code06-01.sxw   bibli : Conditions Module5
Option Explicit

Sub Main5()
Dim an As Long, Resu As Variant

an = 4 ' essayez diverses valeurs
Resu = IIf((an>=0) and (an<100), an+2000, "Année incorrecte")
print Resu
End Sub
```

Choose

La fonction `Choose` est une généralisation de la fonction `IIf`. Au lieu de se limiter à deux alternatives, elle en choisit une parmi plusieurs. La syntaxe de la fonction `Choose` est :

```
Resultat = Choose(expr0, expr1, expr2, expr3, expr4(, etc.))
```

Si l'expression `expr0` vaut 1, `Resultat` reçoit la valeur résultant de `expr1` ; si `expr0` vaut 2, `Resultat` reçoit la valeur résultant de `expr2`, et ainsi de suite.

Si la valeur de `expr0` est inférieure à 1 ou supérieure au nombre d'expressions qui suivent, `Choose` renvoie la valeur `Null`. Incidemment, ceci vous montre un exemple d'utilisation de la pseudo-valeur `Null` d'un `Variant`. Toute tentative d'utilisation d'un tel résultat entraînera une erreur d'exécution. Ici encore, les expressions de la liste peuvent être de différents types.

Le plus souvent, `Choose` sert à choisir un élément dans une liste qui ne sera plus utilisée ailleurs. Prenons pour exemple un tirage de cartes à jouer :

```
rem Code06-01.sxw   bibli : Conditions Module6
Option Explicit

Sub Main6()
Dim tirage As Long, carte As String

'Rnd() fournit un nombre aléatoire entre 0 et 1
tirage = Rnd()*12 +1 ' obtenir un entier de 1 à 13
carte = Choose(tirage, "As", "Deux", "Trois", "Quatre", "Cinq", "Six", "Sept",_
                       "Huit", "Neuf", "Dix", "Valet", "Dame", "Roi")
print "Tirage :" & tirage, carte
End Sub
```

Switch

La fonction `Switch` est encore une autre méthode de choix. Elle n'a aucun rapport avec la fonction du même nom du langage C. La syntaxe est de la forme :

```
Resultat = Switch(expb1, expr1, expb2, expr2, (etc.))
```

Les termes `expb1`, `expb2` désignent des expressions à résultat booléen. Les termes `expr1`, `expr2` désignent des expressions donnant un résultat de type quelconque.

La fonction `Switch` utilise des arguments par paires, avec au moins une paire. L'exécution de la fonction consiste à chercher de gauche à droite la première expression booléenne qui est vraie (`expb1`, puis `expb2`, et ainsi de suite). Dans ce cas, elle renvoie la valeur de l'argument suivant. En cas d'échec, la fonction renvoie la valeur `Null`.

L'exemple suivant utilise la fonction `Switch` pour afficher la couleur correspondant à la synthèse additive des trois couleurs primaires.

```
rem Code06-01.sxw   bibli : Conditions Module7
Option Explicit

Sub Main7()
Dim Rouge As Boolean, Bleu As Boolean, Vert As Boolean
```

```
' essayez diverses valeurs
Rouge = true
Vert = true
Bleu = true
print Switch(Rouge and Vert and Bleu, "Blanc",_
             Rouge and Vert         , "Jaune",_
             Rouge          and Bleu, "Magenta",_
             Rouge                  , "Rouge",_
                       Vert and Bleu, "Cyan",_
                       Vert         , "Vert",_
                                Bleu, "Bleu",_
                                True, "Noir")
End Sub
```

Tout est réalisé dans l'instruction `Print`. Le fonctionnement utilise le fait que `Switch` choisit la chaîne de caractères correspondant à la première expression booléenne vraie.

For Next

Cette boucle permet de répéter un certain nombre de fois les instructions qui sont encadrées par `For` et `Next`. Syntaxe :

```
For index = expr1 To expr2 Step expr3
  instructions
Next
```

La variable `index` est d'un type numérique ; `expr1` et `expr2` sont des expressions numériques qui sont évaluées une seule fois, à l'entrée dans la boucle ; `expr3` est une expression numérique évaluée à chaque tour de la boucle.

La partie `Step` et `expr3` est optionnelle, elle définit l'incrément de la variable `index`. On utilise très couramment pour `expr3` une valeur fixe comme 1 ou -1. L'absence de cette partie équivaut à `Step 1`.

En pratique, on utilise une boucle `For` pour incrémenter ou décrémenter la valeur `index` depuis la valeur `expr1` jusqu'à la valeur `expr2` après chaque tour de boucle. La valeur `index` est initialisée à `expr1`.

Si la valeur `expr2` est supérieure à la valeur `expr1`, il s'agit d'une boucle d'incrémentation. Si `expr3` est négatif, la boucle se termine et l'exécution reprend après la ligne `Next`.

Si la valeur `expr2` est inférieure à la valeur `expr1`, il s'agit d'une boucle de décrémentation. Si `expr3` est positif, la boucle se termine et l'exécution reprend après la ligne `Next`.

Si `expr3` a une valeur nulle, la boucle tournera indéfiniment !

Cet exemple affiche le cube d'une série de nombres réels :

```
rem Code06-01.sxw   bibli : Boucles Module1
Option Explicit

Sub Main1()
Dim x As Double

For x = 37.3 To 63 Step 3.058
  print x, x*x*x
Next
End Sub
```

Remarquez les valeurs successives de `x`, qui ne sont pas nécessairement des entiers.

Les boucles `For` peuvent être imbriquées comme des poupées russes, mais elles ne doivent pas se chevaucher. On peut répéter dans le `Next` le nom de la variable de la boucle, ce qui permet une vérification syntaxique par Basic. Il est recommandé d'indenter les instructions comme dans cet exemple :

```
rem Code06-01.sxw   bibli : Boucles Module2
Option Explicit

Sub Main2()
Dim x As Long, y As Long, z As Long
Dim maTable(5, 7, 11) As Long

For x = 0 to 5
  For y = 0 to 7
    For z = 0 to 11
      maTable(x,y,z) = x*10000 + y*100 +z
    Next z
  Next y
Next x
End Sub
```

L'instruction `Exit For` permet de sortir prématurément d'une boucle `For`. L'exécution reprend après la ligne `Next`.

```
rem Code06-01.sxw   bibli : Boucles Module3
Option Explicit

Sub Main3()
Dim uneTable(200) As String
Dim x As Long, trouve As Boolean

' ici uneTable ne contient que des chaînes vides
uneTable(192) = "OpenOffice" ' remplir un élément
trouve = false
```

```
For x = 0 to 200
  If uneTable(x) = "OpenOffice" Then
    trouve = true
    Exit For
  End If
Next
If trouve Then print x   Else print "Pas trouvé"
End Sub
```

Ainsi que l'indique l'exemple ci-dessus, vous pouvez utiliser la valeur de l'index dans le cas d'une sortie par `Exit For`. En revanche, à la fin normale d'une boucle `For`, la valeur finale de l'index dépend de l'implémentation. Il faut donc utiliser un autre moyen pour distinguer la fin normale et la fin prématurée de la boucle. C'est la fonction de l'indicateur booléen `trouve` dans notre exemple.

Dans le cas de boucles `For` imbriquées, `Exit For` termine la boucle la plus interne.

While Wend

Le terme anglais `While` signifie « Tant que ». Le terme `Wend` peut se traduire par « Fin du Tant que ». La boucle `While` répète une série d'instructions tant qu'une condition est vraie. La syntaxe est :

```
While exprB
  instructions
Wend
```

Au début de chaque tour de boucle, l'expression booléenne `exprB` est évaluée. Si elle est vraie, les instructions encadrées par les lignes `While` et `Wend` sont exécutées, puis l'exécution revient sur la ligne `While`. Si `exprB` est évaluée à faux, la boucle se termine et reprend après la ligne `Wend`. L'exemple suivant est équivalent au premier exemple de boucle `For`.

```
rem Code06-01.sxw   bibli : Boucles Module4
Option Explicit

Sub Main4()
Dim x As Double

x = 37.3
While x<= 63
  print x, x*x*x
  x = x +3.058
Wend
End Sub
```

On voit qu'à la différence de la boucle `For`, le programmeur doit initialiser et incrémenter sa variable lui-même. Un oubli peut entraîner un bouclage sans fin.

Les boucles `While` peuvent être imbriquées, mais elles ne doivent pas se chevaucher.

Il n'est pas possible de sortir prématurément d'une boucle `While`. Utilisez plutôt une boucle `Do` dans ce cas.

Do Loop

La boucle Do peut s'utiliser sous plusieurs formes :

```
' Forme 1
Do While exprB
  instructions
Loop

' Forme 2
Do
  instructions
Loop While exprB

' Forme 3
Do Until exprB
  instructions
Loop

' Forme 4
Do
  instructions
Loop Until exprB

' Forme 5
Do
  instructions
Loop
```

La forme 1 est équivalente à la boucle `While`. La forme 2 exécute au moins une fois les instructions et se répète tant que `exprB` est vraie.

La boucle de la forme 3 se répète jusqu'à (anglais : *until*) ce que `exprB` soit vraie. La forme 3 effectue le test avant chaque exécution des instructions, alors que la forme 4 effectue le test après chaque exécution.

L'instruction `Exit Do` permet de sortir prématurément d'une boucle `Do`. Nous pouvons écrire l'équivalent du deuxième exemple `For` ainsi :

```
rem Code06-01.sxw   bibli : Boucles Module5
Option Explicit

Sub Main5()
Dim uneTable(200) As String
Dim x As Long, trouve As Boolean

' ici uneTable ne contient que des chaînes vides
uneTable(192) = "OpenOffice" ' remplir un élément
x = 0
Do While x <= 200
  If uneTable(x) = "OpenOffice" Then Exit Do
  x = x+1
Loop
If x<=200 Then print x   Else print "Pas trouvé"
End Sub
```

Comme pour la boucle `While`, le programmeur doit veiller à initialiser et incrémenter correctement sa variable principale. Contrairement à la boucle `For`, le programmeur garde une totale maîtrise de sa variable de boucle.

Il existe enfin une cinquième forme de boucle `Do` :

```
' Forme 5
Do
  instructions
Loop
```

La forme 5 est une boucle infinie. Dans ce cas, la terminaison de la boucle est décidée dans une des instructions encadrées par la boucle, et la sortie est effectuée par un `Exit Do`.

GoTo, On Goto

Ces instructions, les plus anciennes de l'histoire de la programmation, ne devraient en principe plus être utilisées que dans dans des contextes particuliers, comme le traitement d'erreur, que nous verrons au chapitre 9.

```
instructions1
GoTo etiquette
instructions2
etiquette:
instructions3
```

L'instruction `GoTo` (en français : Aller à) transfère l'exécution du programme à la ligne commençant par le nom mis en argument de l'instruction `GoTo` et immédiatement suivi par le caractère « : ». Le point d'arrivée doit se situer à l'intérieur de la même `Sub` ou

`Function`. Il peut se situer sur une ligne avant ou après l'instruction `Goto` qui la référence. Il peut même permettre de « sauter » à l'intérieur d'une boucle, ce qui évidemment n'est pas du tout recommandé.

Le nom qui sert d'étiquette suit les mêmes règles de nommage qu'une variable. Une étiquette peut être référencée par plusieurs `GoTo`, mais elle n'existe qu'à un seul endroit. Il est possible de mettre une instruction à droite de l'étiquette sur la ligne d'arrivée.

D'autres langages de programmation disposent d'une instruction `Continue`, qui sert à passer à l'itération suivante dans une boucle. L'instruction `GoTo` permet de simuler cette instruction :

```
For x = 1 to 1000
  - instructions -
  - instructions -
  if condition1 then GoTo Continue1
  - instructions -
  - instructions -
  if condition2 then GoTo Continue1
  - instructions -
  - instructions -
  if condition3 then GoTo Continue1
  - instructions -
  - instructions -
Continue1:
Next x
```

Sans l'utilisation du `GoTo` il faudrait imbriquer des instructions `If` ainsi :

```
For x = 1 to 1000
  - instructions -
  - instructions -
  if not condition1 then
    - instructions -
    - instructions -
    if not condition2 then
      - instructions -
      - instructions -
      if not condition3 then
        - instructions -
        - instructions -
      end if
    end if
  end if
Next x
```

Le `GoTo` est encore parfois utilisé par certains programmeurs pour sortir prématurément d'une boucle. Ce n'est pas nécessaire avec le Basic d'OpenOffice.org, qui dispose pour cela des instructions `Exit For` et `Exit Do` que nous avons vues, et des `Exit Sub` et

`Exit Function` que nous verrons dans le chapitre 7. Ces instructions sont bien préférables, car elles remplissent le même rôle sans produire une « programmation spaghetti ».

L'instruction `On GoTo` permet de poursuivre l'exécution à l'étiquette de même rang que la valeur d'une variable entière. Exemple :

```
rem Code06-01.sxw   bibli : Branchements Module1
Option Explicit

Sub Main1()
Dim n As Long

n = 4
On n GoTo etiq1, etiq2, etiq3, etiq4
print "Pas de saut"
GoTo Fini
etiq1:
  print "Nord"
GoTo Fini
etiq2:
  print "Est"
GoTo Fini
etiq3:
  print "Sud"
GoTo Fini
etiq4:  print "Ouest"
Fini:
End Sub
```

Si la variable `n` vaut 4, l'exécution se poursuit à `etiq4`. Si la variable `n` est inférieure ou égale à zéro, ou supérieure au nombre d'étiquettes en arguments, il n'y a pas de saut et l'exécution se poursuit à l'instruction suivante.

L'exemple montre la multiplication inévitable des `Goto` quand on commence à utiliser ces instructions. Le code devient très difficile à comprendre, car sa structure écrite ne reflète plus son comportement dynamique.

Conclusion

Les instructions de traitements conditionnels et itératifs que nous venons d'exposer sont à la base de tout programme. Elles représentent la logique interne du processus.

Le chapitre suivant nous permettra d'introduire les notions de sous-programme et de fonction qui permettent de hiérarchiser vos développements voire de les réutiliser.

7

Les sous-programmes

Vous apprendrez ici en quoi consiste exactement un sous-programme et quelles sont ses possibilités. OpenOffice.org étant organisé en bibliothèques, vous apprendrez aussi comment utiliser des sous-programmes situés dans une autre bibliothèque.

Sub : le sous-programme

Un sous-programme consiste à « mettre en boîte » tout un codage qui réalise un travail particulier. On peut alors utiliser un simple nom au lieu d'écrire une séquence complète d'instructions. On peut même utiliser un sous-programme écrit par une autre personne, sans se préoccuper de son codage interne : c'est la notion d'abstraction, au sens de « faire abstraction des détails inutiles ». Ce concept est extrêmement puissant et permet de construire progressivement des programmes très sophistiqués.

Le sous-programme le plus courant s'appelle `Sub` et vous l'avez déjà rencontré dans sa forme la plus simple.

```
Sub monSousProgramme
' ici toutes les instructions du sous-programme
' - - - - -
End Sub
```

Le sous-programme de ce codage a pour nom `monSousProgramme`. Il débute par le mot-clé `Sub` et se termine par les deux mots-clés successifs `End Sub`. Cet encadrement doit être réalisé comme indiqué, sur des lignes différentes.

> **ATTENTION Nom des sous-programmes**
>
> Le nom d'un sous-programme doit être unique dans la bibliothèque à laquelle il appartient, car il peut être appelé de n'importe quel module. En particulier, ne pas utiliser plusieurs fois un nom comme `Main`, même dans différents modules.
>
> Dans une version future, il sera possible de déclarer des sous-programmes `Private` à l'instar des données.

Appeler un sous-programme

Un sous-programme est utilisé en l'appelant par son nom, sur une ligne. Suivant le nombre d'arguments, l'appel se présente différemment. Habituellement, la liste des arguments est encadrée de parenthèses, les arguments étant séparés par des virgules.

```
monSousProgramme                ' aucun argument (1ère forme)
monSousProgramme()              ' aucun argument (2ème forme)
monSousProgramme(arg)           ' un argument (1ère forme)
monSousProgramme arg            ' un argument (2ème forme)
monSousProgramme(arg1,arg2,arg3) ' plusieurs arguments (ici 3)
monSousProgramme arg1 arg2 arg3  ' plusieurs arguments (2ème forme)
```

Basic accepte aussi d'utiliser le mot-clé `Call` pour appeler un sous-programme. Pour un débutant, ce peut être un moyen pour distinguer l'appel à un sous-programme de l'utilisation d'un mot-clé. Néanmoins, ce n'est absolument nécessaire que pour l'appel d'une DLL (bibliothèque externe chargée à la demande).

```
Call monSousProgramme(arg)
```

Un sous-programme peut appeler un sous-programme, qui peut lui-même appeler un sous-programme, et ainsi de suite. Il est même possible, dans certaines conditions, que cette suite d'appels forme une boucle (c'est la récursion). À partir de la version 1.1 d'OpenOffice.org, Basic autorise la récursivité, mais dans une limite raisonnable. La récursion est une méthode employée par les programmeurs expérimentés dans certains algorithmes. Il est rare d'en avoir un réel besoin.

L'ordre dans lequel les sous-programmes sont écrits n'a en général pas d'importance : un sous-programme A peut appeler un sous-programme B écrit avant ou après lui. Dans quelques cas, l'analyseur syntaxique de Basic peut cependant mal interpréter vos intentions. Dans nos exemples, nous écrivons en premier le sous-programme principal simplement parce que cela permet de l'exécuter directement en cliquant sur les icônes d'exécution dans l'EDI.

Les paramètres de sous-programme

Pour qu'un sous-programme soit autre chose qu'un simple raccourci, nous avons besoin de lui transmettre un ou plusieurs argument(s) (appelés aussi paramètres), dont la valeur dépendra de nos besoins.

Les paramètres d'un sous-programme sont énumérés dans la ligne `Sub`, comme dans cet exemple :

```
Sub monSousProgramme(arg1 As String, arg2 As Long, arg3 As Long)
```

Le nombre de paramètres est quelconque. Le nom de chacun est également à votre choix ; il servira de variable dans le corps du sous-programme. L'emploi (facultatif) du terme `As` suivi d'un type est identique à celui déjà vu dans l'instruction `Dim`. En effet, nous déclarons ici le type de chaque paramètre du sous-programme.

Voici un sous-programme à paramètres, basé sur l'exemple du chapitre 6 concernant l'instruction `Switch`.

```
rem Code07-01.sxw   bibli : Standard Module1
Option Explicit

Sub EssaisCouleurs()
Dim maCoul As Boolean

SyntheseCouleurs(false, true, true)
maCoul = false
SyntheseCouleurs(true, true, maCoul)
End Sub

Sub SyntheseCouleurs(Rouge As Boolean, Vert As Boolean, _
                     Bleu As Boolean)

print Switch(Rouge and Vert and Bleu, "Blanc",_
             Rouge and Vert         , "Jaune",_
             Rouge          and Bleu, "Magenta",_
             Rouge                  , "Rouge",_
                       Vert and Bleu, "Cyan",_
                       Vert         , "Vert",_
                                Bleu, "Bleu",_
                                True, "Noir")
End Sub
```

Encore une fois, pour des raisons de mise en page, nous avons utilisé des caractères de continuation pour éviter des lignes trop longues (et pour clarifier la structure du codage). Le sous-programme `SyntheseCouleurs` utilise trois paramètres, tous de type `Boolean`, qui sont, dans l'ordre, les indicateurs de couleur rouge, verte, bleue.

Ce sous-programme est appelé deux fois dans le corps du sous-programme `EssaisCouleurs` : une première fois en donnant directement les valeurs des paramètres, une deuxième fois en employant une variable. Dans l'appel d'un sous-programme à paramètres, l'ordre des arguments doit correspondre à l'ordre dans lequel ils sont déclarés dans l'instruction `Sub`.

Paramètres optionnels

Pour simplifier l'usage de certains sous-programmes, il est pratique de rendre optionnel(s) un ou plusieurs paramètre(s). Voici un petit exemple :

```
rem Code07-01.sxw   bibli : Standard Module2
Option Explicit

Sub essaisSalutations()
Saluer("Bonjour", 3)   ' à une demoiselle
Saluer("Bonsoir", 1)   ' à un monsieur
Saluer("Bonne soirée") ' à la cantonade, 2è argument absent
' un appel incorrect, mais accepté par le codage
Saluer("Salut !", -145)
End Sub

Sub Saluer(salut As String, Optional civilite As Long)

if IsMissing(civilite) then civilite = 0
Select Case civilite
Case 1
  print salut & " monsieur"
Case 2
  print salut & " madame"
Case 3
  print salut & " mademoiselle"
Case Else
  print salut
End Select
End Sub
```

Le sous-programme `Saluer` comporte un paramètre déclaré optionnel grâce au mot-clé `Optional`. Ce paramètre prend normalement une valeur entre 0 et 3. Notez que nous avons choisi le dernier paramètre. Si nous avions plusieurs paramètres optionnels, ils devraient être déclarés après tous les paramètres obligatoires. En effet, la règle d'appel est la suivante : si un paramètre est omis, il ne doit plus y avoir d'autre paramètre utilisé à sa droite.

Comme le paramètre `civilite` peut être absent, Basic fournit la fonction `IsMissing` qui renvoie la valeur `true` en cas d'absence, et `false` s'il est présent. S'il est absent, il est tout de même possible de lui affecter une valeur, qui sera utilisée à l'intérieur du sous-programme.

Si le sous-programme comporte plusieurs paramètres optionnels, vous devez utiliser la fonction `IsMissing` sur le premier, puis le second, etc, dans l'ordre de gauche vers la droite ; sinon vous obtiendrez un résultat aberrant.

Le sous-programme `essaisSalutations` effectue différents cas d'appels. Le dernier appel vous montre que le sous-programme `Saluer` est robuste : il fonctionne correctement et donne un résultat raisonnable parce que nous avons utilisé le `Case Else` pour traiter les cas non prévus comme le cas zéro.

Nombre d'arguments transmis au sous-programme

OOoBasic ne contrôle pas le nombre d'arguments passés à un sous-programme, alors que dans la plupart des autres langages de programmation (Java, Python, Pascal...) un nombre incorrect d'arguments est repéré à la compilation. Considérons un programme principal Test qui appelle un sous-programme Truc, déclaré avec trois arguments.

```
Sub Truc(A,B,C)
Msgbox "Truc !"
End Sub

Sub Test
Truc()        ' aucun argument
Truc(1,2)     ' manque un argument
Truc(1,2,3,4) ' argument excédentaire
End Sub
```

Nous avons placé la définition du sous-programme Truc avant celle du programme Test, pour éviter une erreur de syntaxe. En effet, si l'appel de la fonction est placé avant sa définition, Basic considère que la fonction est alors définie par l'intermédiaire du nombre d'arguments passés. Dans notre cas, nous obtiendrions le message « Le symbole Truc a déjà reçu une autre définition » car les appels auraient indiqué un passage de 4 arguments alors que la déclaration `Sub` n'en contient que 3.

Pour notre exemple, nous définissons au préalable notre sous-programme Truc, comportant trois arguments. Exécutez le sous-programme Test : les trois appels à Truc sont réalisés sans erreur.

Vous objecterez que, si Truc possède trois arguments, ils devraient être utilisés et, dans ce cas, une erreur d'exécution apparaîtrait. Oui, mais on peut concevoir un sous-programme qui n'utilise l'argument `C` que pour certaines valeurs de `A` et `B`. Notez que l'usage de la fonction `IsMissing()` est interdite si l'argument testé n'a pas été déclaré `Optional`.

Ce laxisme sur le nombre d'arguments sera utilisé dans les exemples de traitement d'événement que nous verrons aux chapitres 16 et 18 : l'objet événement est toujours transmis au sous-programme de traitement, mais celui-ci peut être déclaré comme n'ayant aucun argument.

Cette fonctionnalité a donc un intérêt, mais soyez vigilants dans la définition et l'utilisation de vos `Sub` et `Functions`.

Transmission par référence ou par valeur

Par défaut, les valeurs des paramètres employés lors de l'appel d'un sous-programme sont transmises par *référence*. Ceci veut dire que, si vous transmettez une variable, celle-ci peut être modifiée par le sous-programme. Voici un exemple artificiel mais démonstratif :

```
rem Code07-01.sxw   bibli : Standard Module3
Option Explicit

Sub essaisTransmission()
Dim p1 as Long
Dim p2 as String

p1 = 123
p2 = "Hello"
print p1, p2 ' afficher les valeurs avant l'appel
Modifieur(p1, p2)
print p1, p2 ' afficher les valeurs après l'appel
End Sub

Sub Modifieur( aa As Long, bb As String)
aa = -4578
bb = "xxxxx"
End Sub
```

Ce comportement est tout à fait utile dans bien des cas. En revanche, il peut être la cause d'erreurs difficiles à repérer si on a modifié la valeur du paramètre par inadvertance. Dans d'autres cas, il est franchement gênant.

Pour éviter de définir dans le sous-programme une variable interne dans le seul but de ne pas modifier le paramètre transmis, Basic offre l'option `ByVal` (abrégé anglais de *By Value*, en français Par Valeur). Modifiez ainsi une des déclarations d'arguments de la macro `Modifieur` :

```
Sub Modifieur( aa As Long, ByVal bb As String)
```

Relancez la macro `essaisTransmission`. La variable transmise ne sera plus modifiée. Nous avons demandé à Basic de passer seulement la valeur du paramètre et nous pouvons alors utiliser le nom `bb` comme une variable interne à la macro.

Les programmeurs puristes utiliseront `ByVal` chaque fois qu'ils n'ont besoin que d'une copie de l'information. Notez cependant que cela induit un (très léger) ralentissement du programme.

Transmettre un tableau dans un paramètre

Il est possible de passer un tableau en paramètre d'un sous-programme : le nom du tableau transmis est alors accompagné de parenthèses vides.

```
Sub testParamTableau()
Dim tata(5,2) As Double
tata(1,0) = 123
tata(2,1) = -5.7
tata(3,2) = pi
manipulerTableau(tata()) ' transmettre un tableau
End Sub


Sub manipulerTableau( v() As Double )
print UBound(v,1), UBound(v,2), v(3,2)
End Sub
```

Le sous-programme manipulerTableau déclare que son argument d'appel est une variable tableau, sans préciser de dimension. Les parenthèses ne sont pas obligatoires, mais recommandées afin de bien illustrer à la relecture le type de données que vous aller manipuler. L'instruction `print` affiche l'index maximal de la première dimension, l'index maximal de la deuxième dimension, et un des éléments du tableau. Dans un codage réel, nous utiliserions les fonctions `LBound()` et `UBound()` pour déterminer les dimensions du tableau avant de traiter ses éléments.

Portée des variables d'un sous-programme

Comme vous avez pu le constater dans quelques exemples, l'instruction `Dim` est utilisable dans un sous-programme. Elle a pour effet de définir des variables qui ne seront reconnues que lorsque Basic exécute des instructions de ce sous-programme. En dehors de ce sous-programme, ces variables n'existent pas, et par conséquent une autre partie du programme peut déclarer les mêmes noms de variables pour son propre usage.

Les noms des arguments du sous-programme sont assimilés à des variables internes, avec les mêmes conséquences. Ce comportement, appelé *localisation des variables*, est extrêmement utile dans des programmes assez conséquents. En effet, le programmeur concevant un sous-programme a ainsi toute liberté pour choisir le nom des variables internes à ce sous-programme. C'est encore un aspect de la modularité.

Cet exemple assez extrême vous démontre l'effet de localisation des variables. Il sera encore plus instructif en l'exécutant avec l'icône « Étape par étape » de l'EDI (voir chapitre 3), car vous verrez la progression du pointeur d'exécution et l'évolution de la pile des appels.

```
rem Code07-01.sxw   bibli : Standard Module4
Option Explicit

Private uneVariable As String ' connue dans tout le module

Sub demoLocalisation()
uneVariable = "privée"
print 1,uneVariable
Dim uneVariable As String ' connue dans demoLocalisation
uneVariable = "demoLocalisation"
print 2,uneVariable
etage1
print 3,uneVariable
End Sub

Sub etage1()
print 11,uneVariable
Dim uneVariable As String ' connue dans etage1
uneVariable = "etage1"
print 12,uneVariable
etage2(uneVariable)
print 13,uneVariable
End Sub

Sub etage2(source As String)
print 21,uneVariable
Dim uneVariable As String ' connue dans etage2
uneVariable = "etage2 avec argument : " & source
print 22,uneVariable
source = source & " modifié"
End Sub
```

Les variables statiques

Les variables déclarées dans un sous-programme sont « neuves » à chaque nouvelle exécution du sous-programme. Autrement dit, elles sont initialisées par la déclaration `Dim` comme nous l'avons vu au chapitre 5.

L'exception à cette règle est la déclaration `Static`. Elle déclare une ou plusieurs variable(s), comme `Dim`. Vous pouvez déclarer des variables tableau, mais les dimensions doivent être fixées à la déclaration et ces tableaux ne peuvent être redimensionnés. Les variables déclarées `Static` sont initialisées au premier lancement du sous-programme, puis conservent leur valeur d'un lancement à l'autre, *jusqu'à la fin de l'exécution* du programme Basic.

Le sous-programme principal `essaiStatique` appelle plusieurs fois le sous-programme `voirTampon` qui affiche la valeur de variables `Static`. En exécutant plusieurs fois ce programme, vous vérifierez que ces variables sont bien réinitialisés à chaque exécution Basic.

```
rem Code07-01.sxw   bibli : Statiques Module1
Option Explicit

Sub essaiStatique()
Do
  voirTampon
Loop Until MsgBox("Recommencer ?", 4) = 7 ' sortir si "Non"
End Sub

Sub voirTampon()
Static Tampon As String, ess(2, 3) As Long
' afficher la valeur actuelle du tampon
MsgBox("Essai " & ess(1, 2) & "    Tampon = " & Tampon)
Tampon = Time ' mettre l'heure courante
ess(1,2) = ess(1,2) +1 ' compter l'essai
End Sub
```

Fin prématurée d'un sous-programme

Un sous-programme se termine lorsque le pointeur d'exécution rencontre la ligne `End Sub`. Dans certains cas, souvent dans un test conditionnel, l'exécution du sous-programme doit se terminer. Pour simplifier et clarifier le codage, Basic offre la possibilité de terminer immédiatement le sous-programme avec l'instruction `Exit Sub`, comme dans cet exemple :

```
rem Code07-01.sxw   bibli : Standard Module5
Option Explicit

Sub Devinette()
Dim reponse As String
Const solution = "27"
Do
  reponse = InputBox("Devinez un nombre entre 1 et 99")
  if reponse = solution then
    MsgBox("Bravo, vous avez trouvé")
    Exit Sub
  end if
  MsgBox("Raté")
Loop Until reponse = 0
End Sub
```

Essayez de faire la même chose sans utiliser `Exit Sub`, vous verrez que ce n'est pas si simple.

Gosub : le sous-programme interne

Basic offre une possibilité limitée de déclarer des sous-programmes internes à un sous-programme. Le sous-programme interne possède un nom, mais ne peut comporter d'arguments d'appel. Il a cependant accès en lecture et écriture aux variables internes du sous-programme principal, ainsi qu'aux paramètres de celui-ci. Un sous-programme interne est défini ainsi :

```
Sub spEnglobant()
' instructions du sous-programme englobant
' - - -
GoSub sousProgInterne' appel
' - - -
Exit Sub ' terminer ici l'exécution principale

' définition du sous-programme interne
sousProgInterne:
' déclarations et instructions
' - - -
Return
End Sub
```

Le sous-programme englobant doit se terminer par un `Exit Sub` derrière lequel on place le ou les sous-programmes internes. Ceux-ci sont appelés par l'instruction `GoSub`, qui transfère l'exécution à l'étiquette indiquée tout en mémorisant le point actuel d'exécution. Le sous-programme interne se termine obligatoirement par une instruction `Return`.

En employant l'instruction `GoSub`, vous définissez un véritable sous-programme interne à votre routine. On ne peut y rentrer que par l'étiquette (le nom suivi du caractère deux-points), et seulement depuis le sous-programme englobant. Un sous-programme interne peut être utilisé plusieurs fois dans votre routine principale, ou même depuis un autre sous-programme interne de la routine principale. Néanmoins, ce n'est pas le seul intérêt.

Les sous-programmes internes peuvent permettre de réduire la complexité d'une routine, en donnant un nom clair à une partie de codage qui a une finalité évidente, et ceci même si vous ne l'utilisez qu'une fois. Vous séparez ainsi cette partie de codage, tout en pouvant utiliser les variables de la routine qui ont déjà été définies. Voyez cet exemple de devinette améliorée, qui utilise un seul sous-programme englobant deux sous-programmes internes.

```
rem Code07-01.sxw   bibli : PetitJeu Module1
Option Explicit

' utilisation de GoSub pour améliorer la modularité
Sub Devinette2
Dim reponse As Long, nbEssais As Long
Const solution = 77

nbEssais = 0
```

```
Do
  GoSub demander_valeur
  if reponse = solution then
    GoSub Afficher_score
    exit do
  elseif reponse = 0 then
    exit do
  else
    print "Perdu, encore un essai ?"
  end if
Loop
print "Jeu terminé"
exit Sub' fin d'exécution de Devinette2

demander_valeur:
dim valTx As String
do
  valTx = InputBox("Devinez un nombre entre 1 et 99")
  if valTx = "" then
    reponse = 0
    exit do
  end if
  reponse = CLng(Left(valTx, 3))
loop until (reponse>=1) and (reponse<=99)
if reponse <> 0 then nbEssais = nbEssais +1
Return' fin de demander_valeur

Afficher_score:
if nbEssais < 10 then
  print "Vous avez un don de télépathie !"
elseif nbEssais < 20 then
  print "Pas mal, vous avez trouvé en "; nbEssais; " essais"
else
  print nbEssais & _
        " essais ! La persévérance est toujours récompensée."
end if
Return' fin de Afficher_score

End Sub ' fin du sous-programme Devinette2
```

Comparez maintenant la lisibilité avec cette autre version rigoureusement équivalente, qui n'utilise pas de sous-programme interne.

```
rem Code07-01.sxw   bibli : PetitJeu Module2
Option Explicit

' Même routine principale, mais sans utiliser de GoSub
Sub Devinette3()
Dim reponse As Long, nbEssais As Long
Const solution = 77
```

```
nbEss2ais = 0
Do
  dim valTx As String
  do
    valTx = InputBox("Devinez un nombre entre 1 et 99")
    if valTx = "" then
      reponse = 0
      exit do
    end if
    reponse = CLng(Left(valTx, 3))
  loop until (reponse>=1) and (reponse<=99)
  if reponse <> 0 then nbEssais = nbEssais +1
  if reponse = solution then
    if nbEssais < 10 then
      print "Vous avez un don de télépathie !"
    elseif nbEssais < 20 then
      print "Pas mal, vous avez trouvé en "; nbEssais; " essais"
    else
      print nbEssais & _
            " essais ! La persévérance est toujours récompensée."
    end if
    exit do
  elseif reponse = 0 then
    exit do
  else
    print "Perdu, encore un essai ?"
  end if
Loop
print "Jeu terminé"
End Sub ' fin du sous-programme Devinette3
```

Dans la première version, vous saisissez la structure générale du programme dans les premières lignes, qui peuvent s'afficher en totalité à l'écran. Et chaque sous-programme interne est suffisamment simple pour être compris rapidement.

En programmation courante, le `GoSub` est peu utilisé en comparaison des `Sub` ordinaires, qui permettent des appels avec arguments et une meilleur modularité. Cependant `GoSub` pallie partiellement l'impossibilité actuelle de définir des `Sub` limités à un seul module.

Function : le sous-programme fonction

Une fonction est un sous-programme qui renvoie une valeur et se déclare :

```
Function nomFonction(arg1 As type1, arg2 As type2) As typeN
' ici toutes les instructions de la fonction
' - - - - -
End Function
```

Notez bien le terme anglais `Function`, avec un *u*. La liste des arguments de la fonction suit les mêmes règles que pour un sous-programme ordinaire. Le mot clé `As` situé après la liste des arguments précise le type de la donnée renvoyée à l'appelant. Le `As` et le type peuvent être absents ; dans ce cas, la fonction renvoie une donnée de type `Variant`.

Toutes les autres fonctionnalités décrites pour les sous-programmes `Sub` sont disponibles avec les sous-programmes `Function`. Rappelons aussi que l'utilisateur d'une fonction peut ignorer le résultat renvoyé en utilisant la `Function` comme une `Sub`.

Reprenons le premier exemple de ce chapitre et créons une fonction qui renvoie la couleur de synthèse sous la forme d'une chaîne de caractères.

```
rem Code07-01.sxw   bibli : Standard Module6
Option Explicit

Sub essaiCouleurs2()
Dim couleur As String
couleur = Synthetiser(true, false, true)
print "La couleur résultante est : " & couleur
End Sub

Function Synthetiser(Rouge As Boolean, Vert As Boolean, _
                     Bleu As Boolean) As String

Synthetiser = Switch(Rouge and Vert and Bleu, "Blanc",_
                     Rouge and Vert         , "Jaune",_
                     Rouge          and Bleu, "Magenta",_
                     Rouge                  , "Rouge",_
                              Vert and Bleu, "Cyan",_
                              Vert         , "Vert",_
                                       Bleu, "Bleu",_
                                       True, "Noir")
End Function
```

À l'intérieur d'une fonction, le nom de celle-ci est utilisé comme une pseudo-variable à laquelle on affecte une valeur. Ici, la fonction `Switch` de Basic renvoie une valeur, qui est utilisée comme résultat de notre fonction. Il est possible d'affecter plusieurs fois le résultat de la fonction, au cours de l'exécution de celle-ci. Seule la dernière affectation sera renvoyée comme résultat de la fonction.

ATTENTION **Pseudo-variable « nom de la fonction »**

Cette pseudo-variable « nom de la fonction » doit seulement être écrite (en lui affectant une valeur), et non lue. Tout autre usage serait confondu avec l'appel de la fonction (qui serait alors un appel récursif). Une méthode sûre consiste à utiliser une variable locale pour calculer le résultat, puis affecter ce résultat à la fonction, de préférence à un seul endroit du codage.

Une erreur courante consiste à terminer une fonction sans lui avoir préalablement affecté un résultat, par exemple dans un des cas d'une alternative (`if then else` ou `Case`). Dans un tel cas, la fonction retourne la valeur par défaut associée à son type de résultat (zéro pour un type numérique, chaîne de caractères vide, ou `Empty` pour un type `Variant`). Il est préférable de commencer la fonction en affectant dès le départ une valeur à son résultat, puis d'affecter une autre valeur selon les conditions d'exécution.

Il n'y a pas de différence de principe entre une fonction Basic et une fonction que vous réalisez vous-même.

Fin prématurée d'une fonction

Comme pour un sous-programme `Sub`, il est possible de terminer une fonction avant que l'exécution n'arrive à la dernière ligne. On emploie pour cela une instruction spécifique :

```
Exit Function
```

N'oubliez pas d'affecter une valeur de retour à votre fonction avant d'exécuter la ligne `Exit Function`.

Sous-programmes et bibliothèques

Nous avons vu au chapitre 3 que les macros sont conservées dans des modules, eux-mêmes conservés dans une bibliothèque. Pour appeler une macro située dans une autre bibliothèque, nous devons ajouter le nom de cette dernière devant le nom de la macro, comme ici :

```
' affiche le nom du produit (OpenOffice.org ou StarOffice, etc.)
' utilisation de la bibliothèque Tools, module Misc
print Tools.GetProductName()
```

On peut utiliser une variable ou constante `Public` d'une autre bibliothèque du même document, en la qualifiant de la même manière.

```
laBibli.laVariable = 3
```

Si vous ne qualifiez pas le nom de la macro ou de la variable avec le nom de sa bibliothèque, l'appel peut fonctionner, mais pas toujours. De plus, en qualifiant le nom, vous évitez les ambiguïtés lorsque le même nom existe ailleurs.

Ces instructions ne sont valides qu'à la condition d'avoir auparavant chargé la bibliothèque correspondante. Les bibliothèques peuvent vous jouer des tours si vous ne prenez

> **CONSEIL** **Renommez !**
>
> Basic vous propose les noms `Library1`, `Library2`, etc. pour une nouvelle bibliothèque, et `Main` pour une nouvelle macro. Si vous gardez ces propositions, vous vous retrouverez avec des conflits de noms, sans que Basic ne vous prévienne. Choisissez donc des noms personnalisés pour vos macros et bibliothèques.

pas la précaution de les charger avant d'utiliser une de leurs macros. En effet, d'une session à l'autre, ou d'un utilisateur à l'autre, vos macros fonctionneront ou non selon que la bibliothèque appelée aura été chargée ou non par une manipulation, parfois involontaire, ou par une autre macro, parfois bien longtemps auparavant.

Ce chargement peut être effectué manuellement, comme indiqué au chapitre 3. En revanche, charger une bibliothèque par macro nécessite des instructions différentes selon la situation. Les instructions permettant d'accéder aux bibliothèques font appel à l'API d'OpenOffice.org, dont la syntaxe est surprenante pour un débutant.

> **VERSION 2.0** **Conteneur soffice**
>
> Rappelons qu'avec la version 2.0 d'OpenOffice.org, le conteneur `soffice` est affiché sur l'interface utilisateur comme deux arborescences : Mes macros (issues du répertoire `user` de l'utilisateur) et Macros OpenOffice.org.(issues du répertoire partagé `share` du poste).

Configurations d'appels

Moyennant éventuellement le chargement de la bibliothèque cible, il est possible d'appeler :

- une macro de `soffice` à partir d'une macro de `soffice`,
- une macro d'un document à partir d'une macro de ce même document,
- une macro de `soffice` à partir d'une macro d'un document.

En revanche, il n'est pas possible d'appeler :

- une macro d'un document à partir d'une macro de `soffice`,
- une macro d'un document à partir d'une macro d'un autre document.

Plus exactement, c'est possible, mais cela relève de l'acrobatie logicielle.

Charger une bibliothèque

Sans instruction ou manipulation particulière, une macro peut appeler toute autre macro de la même bibliothèque, ou de la bibliothèque `Standard` de `soffice`. Si la macro se trouve dans un document, elle peut aussi appeler les macros de la bibliothèque `Standard` de celui-ci.

Si une bibliothèque est protégée par un mot de passe, il reste néanmoins possible, sans connaître ce dernier, de la charger et d'appeler ensuite ses macros. Pour autant, le contenu

de la bibliothèque demeure caché à l'utilisateur. Même les noms de macros ne sont pas affichés.

BOGUE Chargement d'une bibliothèque

Une faute logicielle fait que l'instruction de chargement d'une bibliothèque réinitialise les variables déclarées `Private` ou `Dim` en dehors d'une macro. Si vous avez de telles variables, chargez plutôt les bibliothèques en début de programme. Le défaut devrait être corrigé pour la version 2 d'OpenOffice.org.

Macro initiale dans une bibliothèque de soffice

Copiez les exemples ci-après dans une bibliothèque de `soffice`. Pour charger la bibliothèque `Bibli2` de `soffice`, nous devons exécuter une fois cette instruction :

```
BasicLibraries.LoadLibrary("Bibli2")
```

Respectez la casse dans l'écriture du nom de la bibliothèque. `BasicLibraries` est un objet API dont nous utilisons une fonction appelée `LoadLibrary` (anglais pour : charger bibliothèque).

Il n'y a pas d'inconvénient à charger plusieurs fois la même bibliothèque, mais c'est parfaitement inutile. Comme nous l'avons déjà signalé, il n'existe pas de moyen de « décharger » une bibliothèque, sauf à arrêter l'application OpenOffice.org.

L'instruction précédente déclenchera une erreur s'il n'existe pas de bibliothèque à ce nom dans `soffice`. Nous pouvons vérifier la présence d'une bibliothèque avec la fonction `hasByName` de l'objet `BasicLibraries` :

```
print BasicLibraries.hasByName("Template")
```

La fonction renvoie `True` si la bibliothèque existe, `False` sinon.

L'objet `BasicLibraries` fournit la liste des noms de bibliothèques qu'il connaît, sous la forme d'un tableau de chaînes de caractères appelé `ElementNames`. Nous allons les afficher avec cet exemple, qui doit lui aussi être exécuté dans une bibliothèque de `soffice`.

```
Sub ListerBiblisSoffice()
Dim lesBiblis() As String
Dim x As Long
lesBiblis = BasicLibraries.ElementNames
for x = LBound(lesBiblis) to UBound(lesBiblis)
  print x, lesBiblis(x)
next
End Sub
```

Nous avons ici une déclaration particulière d'un tableau de chaînes de caractères : les parenthèses vides signifient que le nombre d'éléments du tableau sera défini à l'exécution.

Les fonctions Basic `LBound` et `UBound` nous permettent de connaître les valeurs extrêmes de l'index du tableau obtenu.

Macro initiale dans un document ouvert

Le chargement d'une bibliothèque dépendra de sa situation par rapport au document.

Charger une autre bibliothèque du même document

Il suffit d'utiliser l'objet `BasicLibraries` ainsi qu'il a été expliqué plus haut. Toutes les instructions sont identiques.

L'objet `BasicLibraries` connaît les bibliothèques du conteneur en cours. Précédemment, le conteneur était `soffice` ; ici, le conteneur est le document lui-même.

Charger une bibliothèque de soffice

Pour charger une bibliothèque du conteneur `soffice` alors que notre macro se trouve dans un conteneur de document, nous devons utiliser l'objet `GlobalScope.BasicLibraries`. Il suffit de reprendre les instructions précédentes en ajoutant `GlobalScope` devant `BasicLibraries`.

Charger la bibliothèque `Bibli2` de `soffice` :

```
GlobalScope.BasicLibraries.LoadLibrary("Bibli2")
```

Tester l'existence d'une bibliothèque dans `soffice` :

```
print GlobalScope.BasicLibraries.hasByName("Template")
```

Récupérer la liste des noms de bibliothèques de `soffice` :

```
lesBiblis = GlobalScope.BasicLibraries.ElementNames
```

Charger une bibliothèque de soffice au démarrage

Si vous avez souvent besoin d'une bibliothèque de `soffice` (ou de plusieurs), il est très simple de la charger au démarrage d'OpenOffice.org avec une petite macro :

```
Sub ChargerBiblis()
BasicLibraries.LoadLibrary("Tools")
End Sub
```

Insérez cette macro dans un module de la bibliothèque `Standard` de `soffice` qui, elle, est systématiquement chargée.

Associez cette macro sur l'événement « Démarrage de l'application » dans le contexte OpenOffice.org, comme il est expliqué au chapitre 2. Un procédé similaire permettrait de charger une bibliothèque d'un document lors de son chargement.

Conclusion

Les sous-programmes et fonctions sont indispensables à la structuration de votre programme. Sans eux, point de salut ! Ils permettent de « classer » vos développements pour ne pas avoir des centaines de lignes consécutives dont on perd le fil au fur et à mesure. Cette structuration permet également d'envisager de réutiliser certaines parties élémentaires de programmes d'une macro à l'autre.

Le chapitre suivant va nous permettre à présent d'approfondir notre connaissance des instructions OOoBasic.

8

Les principales instructions de traitement

Vous trouverez dans ce chapitre une description de la plupart des instructions Basic que nous n'avons pas encore vues. Lorsque l'aide en ligne est parfaitement suffisante, nous nous contenterons de signaler l'instruction, et il suffira de vous y reporter. Dans d'autres cas, nous la corrigerons ou nous développerons des aspects non décrits ou mal expliqués.

Dans la section consacrée aux fonctions de conversion, nous vous expliquerons en quoi les conversions automatiques de Basic sont souvent utiles et parfois dangereuses.

Lorsque nous utiliserons dans ce chapitre (et ailleurs) l'expression « l'argument est du type xxx », cela signifie que Basic accepte comme argument une variable de ce type, ou une valeur écrite directement, ou toute expression dont l'évaluation donne une valeur de ce type. De plus, Basic effectue automatiquement des conversions de type lorsque c'est nécessaire pour l'évaluation.

Vous découvrirez que Basic offre de nombreuses fonctions. Si vous ne trouvez pas ce que vous cherchez, peut-être pourrez-vous la réaliser en quelques lignes de codage. Si vous avez besoin d'une des fonctions de Calc, le chapitre 12 vous apprendra à la mettre en œuvre dans un programme Basic.

Les chaînes de caractères

Longueur d'une chaîne

La fonction `Len` sert à déterminer la longueur d'une chaîne, en nombre de caractères. La valeur renvoyée est de type `Long`. Tester si une chaîne est nulle peut se faire de deux manières :

```
if chaine = "" then     ' première méthode
if Len(chaine) = 0 then ' deuxième méthode
```

Comparer deux chaînes de caractères

La fonction `StrComp` compare les chaînes de caractères `Text1` et `Text2`. Elle renvoie -1 ou 1 selon que `Text1` est alphabétiquement classé avant `Text2` ou après ; elle renvoie zéro en cas d'identité.

Complément sur la syntaxe

Pour trouver la syntaxe de `StrComp`, taper le nom de la fonction dans l'EDI et appuyer sur la touche F1.

À noter :

- Lorsque le paramètre `Compare` est omis, ceci est équivalent à mettre la valeur 1, donc une comparaison *tenant compte* de la casse.
- D'une manière générale, chaque caractère est « pesé » selon sa valeur donnée par la fonction `ASC`, la comparaison s'effectuant caractère par caractère, de gauche à droite.
- Un caractère chiffre est classé avant tout caractère alphabétique.
- Un caractère majuscule non accentué est classé avant tout caractère minuscule et tout caractère accentué.
- N'importe quel caractère est « plus lourd » que l'absence de caractère : « soleil » est classé avant « soleil » (comportant un espace final).

Rechercher une chaîne dans une autre chaîne

La fonction `InStr` recherche la première apparition de la chaîne `Text2` dans la chaîne `Text1` et renvoie la position du premier caractère concordant. Si `Text2` ne se trouve pas dans `Text1` la fonction renvoie zéro.

Complément sur la syntaxe

Pour trouver la syntaxe de `InStr`, taper le nom de la fonction dans l'EDI et appuyer sur la touche F1.

À noter :

- Le paramètre `Start` doit prendre la valeur 1 pour une recherche à partir du début de la chaîne `Text1`.
- Lorsque le paramètre `Compare` est omis, ceci est équivalent à mettre la valeur 1, donc une comparaison *sans tenir compte* de la casse.
- OpenOffice.org avant la version 2.0 : le paramètre `Start` étant de type `Integer`, une position de départ supérieure à 32 767 doit être codée en valeur négative. La valeur renvoyée est aussi un entier `Integer`, ce qui donne des valeurs négatives si la position trouvée est supérieure à 32 767. L'annexe B fournit un palliatif sur ce point. L'anomalie est corrigée sur la version 2.0.

Le couteau à découper : Mid

La *fonction* `Mid` renvoie une chaîne de caractères qui est un tronçon de la chaîne initiale. La fonction `Mid` possède 2 ou 3 paramètres.

L'*instruction* `Mid` remplace dans une chaîne de caractères un tronçon par une autre chaîne de caractères. L'instruction `Mid` possède toujours 4 paramètres.

Complément sur la syntaxe

Pour trouver la syntaxe de `Mid`, taper le nom de la fonction dans l'EDI et appuyer sur la touche F1.

À noter :

- Les paramètres `Start` et `Length` sont du type `Long` et non `Integer`.
- Si la chaîne de remplacement est plus longue que le tronçon, la partie gauche de la chaîne est utilisée.
- Si la chaîne de remplacement est plus courte que le tronçon, le tronçon complet est remplacé par la chaîne.
- Si la chaîne de remplacement est de longueur nulle, le tronçon est supprimé de la chaîne initiale.

Supprimer des caractères à gauche et à droite

Les fonctions `LTrim`, `RTrim` et `Trim` servent à débarrasser une chaîne de caractères d'espaces inutiles placés respectivement en début, en fin et aux deux bouts de la chaîne. Ces fonctions renvoient la chaîne de caractères résultante.

Les fonctions `Left` et `Right` servent à récupérer la partie gauche, respectivement droite, de la chaîne donnée en argument.

À noter, concernant `Left` et `Right` :

- Si `n` est supérieur à la longueur de la chaîne, la totalité de celle-ci est renvoyée.
- OpenOffice.org avant la version 2.0 : le paramètre `n` de la fonction étant de type `Integer`, il ne peut pas récupérer une longueur supérieure à 32 767. L'annexe B fournit un palliatif sur ce point. L'anomalie est corrigée sur la version 2.0.

Découper, recoller une chaîne

La fonction `Split` découpe une chaîne de caractères en suivant un caractère de délimitation de segments. Elle renvoie un tableau de chaînes de caractères.

La fonction `Join` aboute les éléments d'un tableau de chaînes de caractères en insérant éventuellement un séparateur entre chaque tronçon.

L'exemple de l'aide en ligne étant trop complexe, nous en donnerons un autre. Ce dernier découpe la chaîne `Cible` à chaque caractère point-virgule, et renvoie cinq segments. Une boucle `For` affiche chaque segment. Puis le tableau obtenu est recollé avec un séparateur barre de fraction. Notez bien la manière de déclarer le tableau et de le transmettre à la fonction `Join`.

```
rem Code08-01.sxw   bibli : Standard Module1
Option Explicit

Sub demoSplit()
Dim Cible As String
Dim resu() As String
Dim x As Long

Cible = "alpha;beta;gamma;delta;epsilon;phi"
resu = Split(Cible, ";", 5) ' découpe en 5 segments max
for x = LBound(resu) to UBound(resu)
  print x, resu(x) ' dernier segment : epsilon;phi
next

Cible = Join(resu(), "/")
print Cible
' résultat : alpha/beta/gamma/delta/epsilon;phi
End Sub
```

Créer une chaîne de caractères

La fonction `String` crée une chaîne constituée de la répétition du même caractère. La fonction `Space` est une spécialisation de `String` dans laquelle le caractère répété est un espace. Ces deux fonctions sont surtout utiles pour définir des chaînes très longues, jusqu'à 65 535 caractères.

Notez que le paramètre `n` des fonctions `String` et `Space` est de type `Long`, et non `Integer`.

Les instructions `LSet` et `RSet` servent à cadrer un texte dans une « zone blanche ». Elles ne suivent pas la syntaxe habituelle, mais celle-ci :

```
RSet v1 = v2
```

Dans cette expression, `v1` est une variable de type `String`, `v2` est une variable ou expression du type `String`. Seule la longueur de `v1` est importante, pas son contenu, qui sera perdu. La variable `v1` reçoit la chaîne `v2` cadrée à droite dans cette longueur, et complétée à gauche par des espaces. L'instruction `Lset`, elle, cadre `v2` à gauche et complète à droite par des espaces.

Formater une valeur numérique en chaîne de caractères

La fonction `Format` renvoie une chaîne de caractères constituée à partir d'une donnée numérique et d'une chaîne dite de formatage. Cette fonction est assez complexe et puissante, mais souffre actuellement de multiples erreurs logicielles (notamment lorsque la variable prend une valeur nulle). Nous déconseillons fortement de l'utiliser dans la version actuelle.

La fonction `Format` a été partiellement corrigée dans la version 2.0 d'OpenOffice.org.

Les fonctions numériques

Signe

La fonction `Sgn` renvoie un nombre entier permettant de déterminer si le nombre en argument est négatif, positif, ou nul. L'argument peut être de type entier ou réel.

Fonctions trigonométriques

Les fonctions `Sin`, `Cos` et `Tan` renvoient respectivement la valeur du sinus, du cosinus et de la tangente d'un angle exprimé en radians. La valeur absolue de l'angle doit être inférieure à 2π, sinon le résultat sera incorrect. La fonction `Atn` renvoie l'arc-tangente de l'argument.

Signalons que Basic connaît la constante Pi, avec la précision nécessaire pour les types réels.

Autres fonctions mathématiques

La fonction `Log` renvoie le logarithme népérien de l'argument.

La fonction `Exp` renvoie l'exponentielle (au sens mathématique) de l'argument.

La fonction `Sqr` renvoie la racine carrée de l'argument.

Nombre aléatoire

À chaque appel, la fonction `Rnd` renvoie un nombre pseudo-aléatoire (c'est-à-dire généré mathématiquement pour assurer une distribution des valeurs très proche du hasard) de type `Double`, dont la valeur est comprise entre zéro et un. Une simple multiplication permet ensuite d'ajuster la gamme des valeurs possibles.

L'instruction `Randomize` initialise le générateur pseudo-aléatoire. En donnant un nombre en argument de `Randomize`, le générateur de `Rnd` produira toujours la même série de valeurs pseudo-aléatoires, ce qui peut faciliter la mise au point du programme.

Les fonctions de date et heure

La fonction `Now` renvoie la date-heure actuelle sous la forme d'une donnée de type `Date`.

Les fonctions `Date` et `Time` renvoient respectivement la date et l'heure actuelles. Le résultat est du type `Date`.

La fonction `Timer` renvoie le nombre de secondes écoulées depuis minuit. Le résultat de la fonction ne peut pas être utilisé directement ; on doit l'affecter d'abord à une variable de type `Long`.

La fonction `GetSystemTicks` renvoie un entier `Long` qui est la valeur actuelle d'un compteur incrémenté par le système d'exploitation. La signification de ce compteur dépend de celui-ci.

L'instruction `Wait` effectue une temporisation, sans pour autant bloquer le système d'exploitation. Attention aux réactions de l'utilisateur pendant cette durée ! L'argument de `Wait` est le nombre de millisecondes d'attente. La précision de la temporisation dépend du système d'exploitation.

> **À RETENIR Penser à l'utilisateur**
>
> L'utilisateur s'inquiète rapidement si son programme semble se figer (15 secondes sont pour lui une éternité). Il est alors tenté de taper au clavier ou de cliquer avec la souris « pour débloquer l'application ». Le résultat peut être catastrophique si votre programme effectuait un traitement un peu long. Vous trouverez au chapitre 19 quelques méthodes pour éviter ce problème.

Les fonctions de conversion

Les données étant de différents types, Basic nous permet de passer d'un type à l'autre grâce à des fonctions de conversions implicites ou explicites.

Les conversions automatiques de type

Dans le but de simplifier la programmation, Basic effectue automatiquement des conversions de type. Ces conversions ont lieu lorsqu'une variable ou une expression d'un type A est affectée à une variable de type B, ou employée comme paramètre d'un argument de type B. Elle peuvent aussi avoir lieu lors de l'évaluation d'une expression mathématique, et cela plusieurs fois.

Basic essaie toujours de convertir, quels que soient les types simples impliqués. Certaines conversions ne posent aucun problème car le type final accepte toutes les valeurs possibles du type initial :

- type `Integer` vers `Long`,
- type `Single` vers `Double`,
- type `Integer`, `Long`, vers `Currency`,
- type simple vers `Variant`.

Quelques pertes apparaissent avec les arrondis nécessaires pour convertir un réel en entier. En revanche, les résultats deviennent surprenants avec les types `String` et `Boolean`.

Type Boolean vers type numérique

La valeur `True` donne le résultat -1, la valeur `False` donne le résultat zéro.

Type Boolean vers type String

On obtient la chaîne de caractères `"True"` ou `"False"`.

Type numérique vers type Boolean

La valeur zéro donne un résultat `False`, toute autre valeur donne un résultat `True`.

Type numérique vers type String

On obtient une chaîne de caractères représentant la valeur décimale de la variable numérique (éventuellement après normalisation, dans le cas d'un type `Single` ou `Double`).

Type String vers type numérique

Si la chaîne de caractères est interprétable comme un nombre, alors ce nombre est converti dans le type final ; sinon le résultat est zéro. Basic recherche le nombre en balayant la chaîne de gauche à droite, ce qui donne par exemple :

```
dim s1 as string, d1 as double
s1 = "    15.3xyz        "
d1 = s1
print d1 ' résultat : 15.3
```

Type String vers type Boolean

Si la chaîne de caractères s'évalue comme un nombre, le traitement est identique à celui d'un nombre ; si la chaîne de caractères est de longueur nulle, ou ne comporte que des espaces, le résultat est `False` ; dans les autres cas, une erreur est déclenchée.

L'ambiguïté de l'opérateur +

La plus grande difficulté survient avec l'opérateur + qui est l'opérateur d'addition de nombres, mais aussi autorisé comme opérateur de concaténation de chaînes. Nous vous avons conseillé d'utiliser l'opérateur & pour concaténer les chaînes, et nous allons voir pourquoi dans cette succession de tests de conversions :

```
rem Code08-02.sxw   bibli : Standard Module1
Option Explicit

' danger des conversions implicites
' et de l'opérateur + à double usage
Sub dangerConversions()
Dim d1 As Long, d2 as Long
Dim s1 As String, s2 As String, s3 As String, s4 As String

s1 = "Victor "
s2 = "Hugo"
s3 = s1 & s2

' Le + est interprété comme opérateur de concaténation
s4 = s1 + s2
print "test1", s3, s4 ' mêmes résultats, opérations identiques

d1 = 400
d2 = 1000
s3 = d1 & d2
' Le + est interprété comme opérateur arithmétique
s4 = d1 + d2
print "test2", s3, s4 ' résultats différents !

' Basic interprète l'expression comme une chaîne de caractères
s4 = "" + d1 + d2
print "test3", s3, s4 ' résultats identiques !
```

```
' Basic interprète d'abord en numérique et finit en chaîne
s4 = d1 + d2 +""
print "test4", s3, s4 ' résultats différents !

s1 = "100"
s2 = "2000"
d1 = s1 + s2 ' attention : concaténation de chaînes
d2 = CLng(s1) + CLng(s2) ' addition de deux nombres
print "test5", d1, d2 'résultats différents !
End Sub
```

Le dernier test montre l'intérêt des conversions explicites, que nous listerons maintenant.

Les conversions explicites

Chaîne de caractères vers nombre

La fonction `Val` convertit une chaîne de caractères représentant un nombre en une valeur numérique de type `Double`. La chaîne de caractères doit être un nombre exprimé sous la forme anglo-saxonne, avec une virgule ou un espace comme séparateur de milliers éventuel, et un point comme séparateur décimal.

La fonction `Asc` convertit le *premier* caractère d'une chaîne en sa valeur numérique ASCII ou Unicode 16 bits.

```
print Asc("A"), Asc("a"), Asc("à"), Asc("â"), Asc("€")
'          65        97       224       226       8364
```

OpenOffice.org avant la version 2.0 : la fonction `Asc` renvoie une valeur de type `Integer`, qui n'est pas adaptée aux caractères Unicode dont la valeur dépasse 32 767. L'annexe B fournit un palliatif sur ce point. L'anomalie est corrigée sur la version 2.0.

Type numérique vers chaîne de caractères

La fonction `Chr` est la transformation inverse de la fonction `Asc`. Elle renvoie une chaîne d'un caractère dont la valeur ASCII ou Unicode correspond à la valeur numérique fournie en argument. Notez que cet argument est du type `Long`, et non `Integer` ; toute la gamme de valeurs entre 0 et 65 535 est utilisable. Certaines valeurs de caractères non imprimables sont particulièrement utiles :

- `Chr(13)` correspond au caractère CR (anglais pour Retour de Chariot), souvent utilisé pour afficher des messages sur plusieurs lignes.
- `Chr(10)` correspond au caractère LF (anglais pour Nouvelle Ligne).
- `Chr(9)` correspond au caractère de tabulation.

La fonction `Str` renvoie une chaîne de caractères correspondant à la valeur numérique en argument. Le format est le plus simple possible.

La fonction `CStr` est plus générale que `Str`, l'argument pouvant être un type quelconque.

Les fonctions `Hex` et `Oct` renvoient une chaîne de caractères correspondant respectivement au codage Hexadécimal ou Octal de la valeur numérique en argument. À propos, voici comment on écrit en Basic un nombre en valeur hexadécimale et en valeur octale :

```
Dim n As Long
n = &HFF2        ' valeur hexadécimale FF2
print n, hex(n)  ' Vérification
n = &O752        ' valeur octale 752
print n, oct(n)  ' Vérification
```

Dans l'expression octale le nombre est précédé de la lettre O, et non pas du chiffre zéro.

Conversions vers un type numérique

Les fonctions suivantes convertissent une chaîne de caractères ou une valeur numérique en un type numérique spécifique :

- `CDbl` renvoie un `Double`. Contrairement à la fonction `Val`, `CDbl` tient compte de la configuration locale. Sur une version française, le nombre devra employer un point ou un espace comme séparateur de milliers éventuel, et une virgule comme séparateur décimal.
- `CSng` renvoie un `Single`.
- `CInt` renvoie un `Integer`.
- `CLng` renvoie un `Long`.
- `CCur` renvoie un `Currency`.

Conversion d'un nombre réel vers un nombre entier

Les fonctions `Int` et `Fix` renvoient un nombre `Double` dont la partie fractionnaire est nulle. La fonction `Fix` renvoie un nombre dont la partie entière correspond à la partie entière du nombre en argument. La fonction `Int` renvoie un nombre arrondi à l'entier inférieur, que l'argument soit positif ou négatif.

Ces deux fonctions ne correspondent pas à l'arrondi classique. En revanche, la fonction `CLng` et la fonction `CInt` arrondissent au nombre entier le plus proche. Elle diffèrent seulement par le type du résultat, respectivement `Long` et `Integer`, et donc la plage de valeurs admissibles en argument.

Ce codage compare les fonctions sur différentes valeurs typiques :

```
rem Code08-01.sxw   bibli : Standard Module2
Option Explicit

Sub PartieEntiere()
'              Résultat - - >     Fix     Int    CLng
comparer( 0)                  '      0       0       0
comparer( 0.00000000000001) '      0       0       0
comparer(-0.00000000000001) '      0      -1       0
comparer( 7.00000000000001) '      7       7       7
comparer(-7.00000000000001) '     -7      -8      -7
comparer( 7.49999999999999) '      7       7       7
comparer( 7.5               ) '      7       7       8
comparer(-7.49999999999999) '     -7      -8      -7
comparer(-7.5               ) '     -7      -8      -8
comparer( 7.99999999999999) '      7       7       8
comparer(-7.99999999999999) '     -7      -8      -8
End Sub
Sub comparer(n As Double)
Dim message As String
message = "Nombre : " & n & chr(13) & _
          "Fix : " & Fix(n) & chr(13) & _
          "Int : " & Int(n) & chr(13) & _
          "CLng : " & CLng(n)
MsgBox(message)
End Sub
```

Valeur absolue d'un nombre

La fonction `Abs` renvoie un nombre `Double` qui est la valeur absolue du nombre donné en argument. L'argument peut être un nombre entier ou un nombre réel.

Changement de casse

Le changement de casse consiste à transformer un caractère majuscule en caractère minuscule ou l'inverse.

- La fonction `Lcase`, pour *Lower Case*, renvoie une chaîne de caractères copiée de celle en argument, mais dont chaque caractère majuscule est remplacé par l'équivalent minuscule. Les autres caractères sont simplement copiés.
- De même, `Ucase`, pour *Upper Case*, effectue la mise en majuscules des caractères minuscules.

Ces conversions traitent correctement les caractères accentués (à devient À).

Conversions de date et heure

Comme nous l'avons vu au chapitre 5, les dates (avec l'heure) sont codées en interne sous forme d'un nombre `Double` appelé dans le jargon OpenOffice.org : *serial number* (anglais pour numéro de série). Plusieurs fonctions effectuent des conversions entre cette notation interne et les valeurs habituelles de date et heure.

Fonctions renvoyant une date-heure interne

- `CDate`(chaîne) : l'argument est une chaîne de caractères représentant une date et/ou une heure.
- `CDate`(nombre) : dans ce cas, l'argument est un nombre réel qui sera interprété comme une date interne.
- `DateSerial`(année, mois, jour).
- `TimeSerial`(heure, minute, seconde).
- `DateValue`(chaîne) : l'argument est une chaîne de caractères représentant une date ; en fait le résultat est de type `Long` car les heures-minutes-secondes sont ignorées.
- `TimeValue`(chaîne) : l'argument est une chaîne de caractères représentant une heure précise.
- `CDateFromIso`(chaîne) : l'argument est une date de la forme année, mois, jour, comme "20041231" ou "2002/12/31".

Fonctions prenant pour argument une date-heure interne

- `Day`(date) renvoie le numéro du jour dans le mois.
- `WeekDay`(date) renvoie le rang du jour dans la semaine : dimanche=1, Lundi=2 , jusqu'à Samedi=7.
- `Month`(date) renvoie le numéro du mois.
- `Year`(date) renvoie l'année.
- `Hour`(date) renvoie l'heure ; voir note.
- `Minute`(date) renvoie le nombre de minutes dans l'heure ; voir note.
- `Second`(date) renvoie le nombre de secondes dans la minute ; voir note.
- `CDateToIso`(date) renvoie une chaîne de 8 caractères comme "20041231".

> **NOTE**
>
> OpenOffice.org avant la version 1.1.4 : les fonctions `Hour`, `Minute` et `Second` renvoient un résultat incorrect pour certaines valeurs. L'annexe B fournit un palliatif sur ce point. L'anomalie est corrigée à partir de la version 1.1.4 et bien sûr 2.0.0.

Conversion vers une valeur booléenne

La fonction `CBool` reçoit en argument une expression booléenne ou un nombre et renvoie une valeur booléenne.

Test de contenu de variable

Certaines de ces fonctions, listées au tableau 8-1, ont déjà été vues au cours du chapitre 5. Toutes prennent en argument une variable et renvoient la valeur booléenne `True` si le résultat du test est positif.

Tableau 8–1 Fonctions de test de contenu de variable

Fonction	Test
`IsArray`	Est-ce un tableau ?
`IsDate`	Est-ce une date valide ?
`IsEmpty`	La variable `Variant` a-t-elle la valeur `Empty` ?
`IsMissing`	L'argument du sous-programe est-il absent dans son appel ?
`IsNull`	La variable `Variant` ou `Object` a-t-elle la valeur `Null` ?
`IsNumeric`	La chaîne de caractères en argument est-elle interprétable comme un nombre ?
`IsObject`	Est-ce un objet OLE ?
`IsUnoStruct`	La chaîne de caractères en argument est-elle interprétable comme un type Uno ?
`EqualUnoObjects(v1, v2)`	Les deux variables v1 et v2 représentent-elles le même objet Uno ?

Interface utilisateur : écran, clavier

MsgBox

Cette fonction, qui possède plusieurs paramètres optionnels, sert à afficher un texte dans une boîte de dialogue standard, accompagné d'un ou plusieurs bouton(s) permettant à l'utilisateur de fermer la fenêtre de message. Elle est très souvent utilisée car elle vous permet d'afficher une boîte de dialogue simple tout à fait équivalente à celles produites par des applications professionnelles.

Dans sa forme la plus simple, elle affiche un message avec un bouton OK et on n'utilise alors pas le résultat renvoyé :

```
MsgBox("Programme terminé")
```

L'aide en ligne ne décrivant pas exactement les possibilités, nous reproduisons ici celles qui sont effectivement utiles.

La fonction `MsgBox` utilise trois paramètres :

1. le texte du message (chaîne de caractères),
2. le type du message (optionnel, entier `Integer`),
3. le titre de la boîte (optionnel, chaîne de caractères) ; si cet argument est utilisé, le type doit aussi être utilisé.

Le texte du message peut s'afficher en plusieurs lignes en employant le caractère « Retour de Chariot » pour changer de ligne, par exemple :

```
dim s1 as string
s1 = "Ceci est un message" & chr(13) & _
     "en plusieurs" & chr(13) & "lignes"
MsgBox(s1)
```

Le type du message est la somme de trois valeurs de choix :

1. le choix de l'ensemble de boutons (tableau 8-2),
2. le choix d'une icône normalisée (tableau 8-3), accompagnée du signal sonore qui lui est affecté par le système d'exploitation,
3. le choix du bouton par défaut (tableau 8-4).

Le résultat de la fonction (tableau 8-5) permet de déterminer quel bouton a été actionné. Le tableau 8-2 indique aussi l'action résultant de la fermeture de la boîte de dialogue (case X dans le coin de la fenêtre).

> ATTENTION **Évolution de MsgBox**
>
> Avant la version 1.1.4 d'OpenOffice.org, la ligne Valeur=2 du tableau 8-2 est identique à la ligne Valeur=5.

Tableau 8–2 Choix de l'ensemble de boutons

Valeur	Boutons			Fermeture boîte
0	OK			OK
1	OK	Annuler		Annuler
2	Interrompre	Réessayer	Ignorer	(Inactif)
3	Oui	Non	Annuler	Annuler
4	Oui	Non		Non
5	Réessayer	Annuler		Annuler

Tableau 8–3 Choix de l'icône

Valeur	Icône	Signification
0	aucune	
16	X	Message critique
32	?	Question
48	!	Avertissement
64	i	Information

Tableau 8–4 Choix du bouton par défaut

Valeur	Rang, de gauche à droite
0	le premier
256	le deuxième
512	le dernier

Tableau 8–5 Résultat renvoyé par MsgBox

Valeur	Signification
1	OK
2	Annuler
3	Interrompre
4	Réessayer
5	Ignorer
6	Oui
7	Non

Print

Le nom de cette instruction est impropre car elle ne sert pas à imprimer. Elle est souvent utilisée pour afficher un message servant à la mise au point des macros. En effet, le panneau comporte les boutons OK et Annuler ; l'activation du bouton Annuler (ou la fermeture de la boîte) avorte l'exécution de la macro, ce qui peut être très utile pour se sortir d'une « boucle infernale ».

Si le message comporte des caractères « Retour de Chariot », Basic traite chaque ligne du message comme un appel à `Print`.

L'instruction `Print` accepte plusieurs arguments avec diverses possibilités d'affichage expliquées dans l'aide en ligne.

InputBox

L'instruction `InputBox` est un moyen simple et rustique de demander une information à l'utilisateur. Cette information est récupérée sous forme d'une chaîne de caractères. Le texte affiché peut comporter jusqu'à trois lignes, en utilisant `chr(13)` comme retour à la ligne.

Il n'est pas possible de savoir si l'utilisateur a annulé la boîte de dialogue ou s'il a déclenché le bouton `OK`. Les dimensions de la boîte sont fixes, ce qui n'est pas vraiment esthétique. Pour avoir plus de possibilités, il faut utiliser une véritable boîte de dialogue, comme nous l'expliquerons au chapitre 15.

Veuillez consulter l'aide en ligne pour les divers paramètres. Notez cependant qu'elle est incorrecte concernant le positionnement : il n'est pas absolu, mais relatif au coin haut/gauche de la fenêtre OpenOffice courante ; par défaut, la boîte est centrée dans cette fenêtre.

Codage des couleurs

Beaucoup d'éléments sont colorés, comme des caractères de texte, le fond d'une cellule, le contour et le fond des dessins. Nous apprendrons dans d'autres chapitres à les manipuler et à changer leur couleur.

OpenOffice.org gère les couleurs sur 24 bits, ce qui nous offre 16 777 216 nuances possibles, du noir au blanc en passant par tout l'arc-en-ciel avec de multiples luminosités. Une couleur est définie à partir des trois couleurs fondamentales : Rouge, Vert, Bleu. L'intensité de chacune d'entre elles est définie par un nombre sur 8 bits, donnant une plage de valeurs de 0 à 255 ; l'absence de cette couleur correspond à la valeur nulle, et l'intensité augmente avec la valeur.

Une couleur particulière peut être mémorisée dans une variable de type `Long`. La fonction `RGB` renvoie la valeur de couleur correspondant aux trois couleurs fondamentales qui la constituent, dans l'ordre : Rouge (anglais *Red*), Vert (anglais *Green*), Bleu (anglais *Blue*).

```
Dim couleur As Long
couleur = RGB(100, 220, 187)
'             rouge vert bleu
```

Vous pouvez visualiser la nuance correspondante en utilisant la palette de couleurs disponible par le menu `Outils > Options > OpenOffice.org > Couleurs` (figure 8-1, malheureusement en noir et blanc).

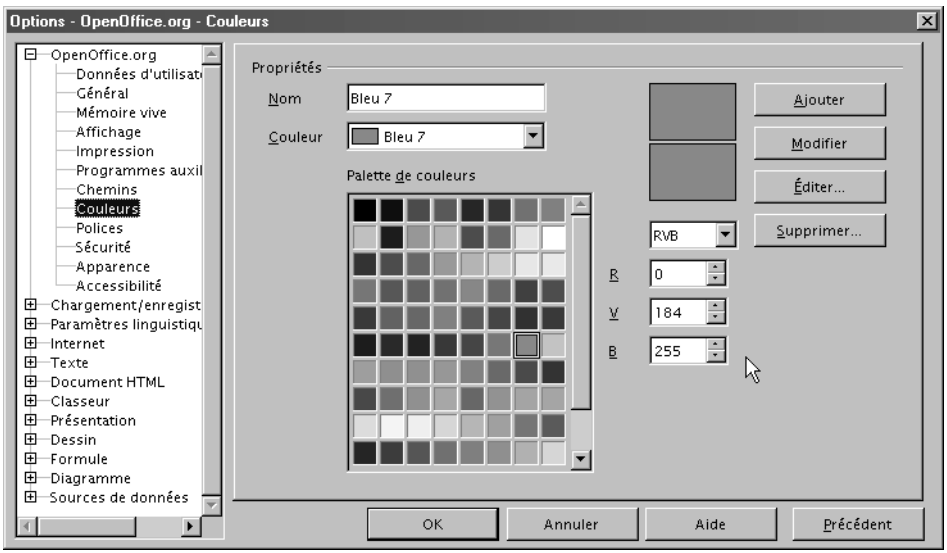


Figure 8–1
La palette de couleurs

Chaque fonction `Red`, `Green`, `Blue` extrait une des couleurs fondamentales constituant une nuance de couleur. Une couleur fondamentale peut être mémorisée dans une variable `Integer` (ou `Long` a fortiori).

```
print Red(couleur), Green(couleur), Blue(couleur)
```

Signalons que le chapitre 19 montre comment, grâce à l'API, vous pouvez connaître les couleurs disponibles dans la palette gérée par OpenOffice.org.

Traitement des fichiers

Syntaxe des adresses de fichiers

La syntaxe des adresses de fichiers ou de répertoires diffère selon les systèmes d'exploitation. Essentiellement, vous aurez à traiter des adresses au format MS-DOS et des adresses au format URL. En effet, même si vous développez vos macros sous Linux, l'utilisateur peut très bien disposer d'un ordinateur Mac OS ou fonctionnant sous MS-Windows.

La fonction `ConvertToURL` renvoie une adresse URL équivalente à l'adresse en format natif du système d'exploitation sur lequel fonctionne votre programme. La conversion inverse est effectuée par `ConvertFromURL`.

L'instruction `getPathSeparator`, non décrite par l'aide en ligne mais utilisée dans des exemples, renvoie le caractère séparateur propre au système d'exploitation utilisé, soit \ pour MS-Windows et / pour les systèmes Unix. Ceci vous permet d'ajouter facilement un nom de fichier à un chemin de répertoire.

```
adrFichier = chemin & getPathSeparator & "toto.bin"
```

Gestion de fichiers

Ces instructions et fonctions s'appliquent sur des fichiers qui ne sont pas ouverts.

Les instructions ou fonctions suivantes ne posent pas de problème particulier d'utilisation et sont décrites clairement dans l'aide en ligne :

- `MkDir` crée un répertoire.
- `RmDir` supprime un répertoire.
- `FileExists` renvoie `True` si un fichier ou un répertoire existe.
- `FileDateTime` renvoie la date-heure du fichier.
- `FileLen` renvoie la taille du fichier.
- `FileCopy` effectue une copie du fichier.
- `Name` renomme le fichier ou le répertoire.
- `Kill` détruit le fichier ou le répertoire.

Les fonctions `ChDrive`, `CurDir` et `ChDir`, bien que documentées, ne sont plus prises en charge. Elles n'ont plus d'utilité dans un système d'exploitation moderne. Le chapitre 19 décrit d'autres moyens pour connaître les répertoires importants du système.

Abordons maintenant les instructions plus complexes.

Explorer un répertoire

La fonction `Dir` sert à explorer un répertoire pour trouver les sous-répertoires ou fichiers qu'il contient. La description de cette fonction dans l'aide en ligne est correcte, mais l'exemple contient des erreurs.

Voici un premier exemple qui liste les sous-répertoires d'un répertoire donné.

```
rem Code08-03.sxw   bibli : Standard Module1
Option Explicit

Sub ListerSousRepertoires1Repertoire()
Dim RepBase As String, unRep As String
```

```
' terminer le nom du répertoire avec le séparateur
RepBase = InputBox("Répertoire à explorer")
unRep = Dir(RepBase, 16)
Do While Len(unRep) > 0
  print unRep
  unRep = Dir' sous-répertoire suivant
Loop
End Sub
```

Le chemin du répertoire est écrit avec la syntaxe du système d'exploitation ou avec la syntaxe URL. Il doit se terminer par un séparateur (caractère \ pour MS-Windows, caractère / pour Linux et syntaxe URL). Vous remarquerez l'existence des pseudo-répertoires « . » et « .. » qui représentent respectivement le répertoire lui-même et le répertoire parent. En général, ces deux répertoires doivent être ignorés avec un test `if`.

Le deuxième exemple liste les fichiers ordinaires qui se trouvent dans un répertoire donné.

```
rem CodeO8-03.sxw   bibli : Standard Module2
Option Explicit

Sub ListerFichiers1Repertoire()
Dim RepBase As String, unFich As String
' terminer le nom du répertoire avec le séparateur
RepBase = InputBox("Répertoire à explorer")
unFich = Dir(RepBase & "*", 0) ' chercher tous les fichiers
Do While Len(unFich) > 0
  print unFich ' utilisez le bouton Annuler pour stopper !
  unFich = Dir' fichier suivant
Loop
End Sub
```

En changeant la chaîne de recherche, vous pourriez par exemple rechercher tous les fichiers d'une extension particulière.

Explorer un arbre de répertoires

Pour explorer un arbre contenant des sous-répertoires imbriqués et des fichiers dans chacun d'eux, il est quasiment indispensable d'employer la récursivité. Cette méthode est permise à partir de la version 1.1 d'OpenOffice.org, mais nous allons avoir une difficulté, car la fonction `Dir` n'est pas utilisable de manière récursive.

Le code suivant présente la méthode générale. Pour l'exemple, nous nous contenterons d'afficher chaque fichier et chaque sous-répertoire repéré.

```
rem CodeO8-03.sxw   bibli : Standard Module3
Option Explicit

Sub ExplorationRecursive()
Dim URLracine As String
```

```
URLracine = convertToURL("c:\")
explorerFichiers(URLracine)
End Sub

Sub explorerFichiers(ceRepertoire As String)
Dim f2 As String

f2 = Dir(ceRepertoire & "*", 0) ' chercher tous les fichiers
Do While Len(f2) > 0
  print ceRepertoire & f2
  f2 = Dir ' fichier suivant
Loop
' appel récursif !
explorerRepertoires(ceRepertoire)
End Sub

Sub explorerRepertoires(ceRepertoire As String)
Dim d2 As String, x As Long
Dim nouvRep As String, listeRep As String

d2 = Dir(ceRepertoire, 16) ' chercher les sous-répertoires
' mémoriser les sous-répertoires car Dir n'est pas récursif
Do While Len(d2) > 0
  if (d2 <> ".") and (d2 <> "..") then
    listeRep = listeRep & d2 & "*"
  end if
  d2 = Dir ' sous-répertoire suivant
Loop
x = 1
d2 = elmtRepSuiv(listeRep, x)
Do While Len(d2) > 0
   nouvRep = ceRepertoire & d2 & "/"
   print nouvRep
   ' appel récursif !
   explorerFichiers(nouvRep)
   d2 = elmtRepSuiv(listeRep, x)
Loop
End Sub

' renvoie l'élément suivant et modifie debut
Function elmtRepSuiv(listElements As String, debut As Long)
Dim x2 As Long

x2 = InStr(debut, listElements, "*")
if x2 = 0 then
  elmtRepSuiv = ""
else
  elmtRepSuiv = Mid(listElements, debut, x2 -debut)
  debut = x2 +1
end if
End Function
```

Initialisez la variable `URLracine` avec le nom d'une arborescence intéressante, et n'oubliez pas de mettre un `\` à la fin.

La macro `explorerFichiers` recherche tous les fichiers du répertoire en argument. Ici, l'utilisation de `Dir` est identique à ce que nous avons vu. Quand tous les fichiers ont été trouvés, la macro lance l'exploration des sous-répertoires du répertoire courant.

Dans la macro `explorerRepertoires`, nous sommes obligés de recourir à une astuce ; en effet, pour chaque sous-répertoire que nous trouverons, nous allons lancer la recherche de fichiers. Comme cette dernière utilise aussi la fonction `Dir`, nous ne pouvons pas être alors en train d'exécuter une recherche avec `Dir`. Nous allons donc d'abord rechercher tous les sous-répertoires et les mémoriser.

Pour effectuer la mémorisation des noms de sous-répertoires, nous n'utilisons pas un tableau de `String`, car il faudrait le redimensionner en cours de recherche avec l'option `Preserve`, ce qui ralentirait l'algorithme. Nous avons préféré mémoriser tous les noms dans un seul `String`, en les séparant par un caractère * qui ne peut être utilisé dans un nom de sous-répertoire. Nous avons de la place, car une chaîne peut contenir 65 535 caractères. En explorant le répertoire `c:\` d'un disque MS-Windows bien encombré, la variable `listeRep` n'a pas dépassé 3,5 % de la taille maximale possible.

Nous récupérons ensuite ces noms un par un avec la fonction `elmtRepSuiv`. Pour chaque nom, après affichage, nous lançons la recherche des fichiers de ce sous-répertoire : ici commence la récursion. Elle se terminera avec l'épuisement des sous-répertoires de chaque sous-répertoire.

Lire, modifier des attributs de fichier ou répertoire

La fonction `GetAttr` sert à lire les attributs d'un fichier ou d'un répertoire. L'instruction `SetAttr` permet de les modifier.

Les attributs indiqués dans l'aide en ligne ne sont pas tous utilisables. Seules ces valeurs ont pu être vérifiées sous MS-Windows XP :

- 0 : fichier sans attribut particulier ;
- 1 : fichier en lecture seule ;
- 2 : fichier caché ;
- 3 : fichier caché et en lecture seule ;
- 16 : répertoire sans attribut particulier ;
- 18 : répertoire caché.

Écrire, lire un fichier

Lire ou écrire un fichier se fait toujours en trois étapes :

1. ouvrir le fichier grâce à l'instruction `Open` ;

2 écrire avec `Print` ou `Write` ou `Put`, ou lire avec `Get` ou `Input` ;

3 fermer le fichier avec `Close`.

L'aide en ligne est malheureusement assez obscure dans ses descriptions et exemples. Nous allons essayer de clarifier ces concepts.

Fichier texte

Le codage suivant écrit un fichier en texte simple (sans formatage).

```
rem Code08-03.sxw   bibli : LireEcrire Module1
Option Explicit

' adaptez le nom et le répertoire à votre système
Public Const nomFichier = "C:\Docs OpenOffice\essai001.txt"

Sub EcrireFichierTexteV1()
Dim f1 As Integer
Dim OuiNon As Boolean

OuiNon = True
f1 = FreeFile ' obtenir un numéro de fichier ouvert
Open nomFichier For Output As #f1
Print #f1, "Ceci est un texte."
Print #f1                      ' ligne vide
Print #f1, "Troisième ligne du texte ", 12345, OuiNon
Close #f1
End Sub
```

Nous définissons une constante contenant le nom et le chemin d'accès au fichier à écrire. Cette constante est publique car nous l'utiliserons dans d'autres modules. Nous aurions aussi bien pu employer une variable. Pour écrire ou lire un fichier, nous avons besoin d'un « numéro de canal d'entrée/sortie » qui nous est renvoyé par la fonction `FreeFile`. Il servira ensuite à préciser à Basic le fichier concerné par chaque instruction d'entrée/sortie (car nous pourrions ouvrir plusieurs fichiers simultanément).

L'instruction `Open` comporte de multiples variantes pour ses arguments. Ici, nous demandons d'ouvrir en écriture (anglais : *For Output*) le fichier dont le chemin d'accès est indiqué par la variable `monFichier` ; nous précisons notre numéro de canal, qui doit être écrit précédé du caractère dièse. L'argument `For Output` provoque l'effacement et la réécriture d'un fichier pré-existant du même nom, sans aucun avertissement.

Les instructions suivantes écrivent chacune une ligne de texte dans le fichier. Nous utilisons l'instruction bien connue `Print`, mais avec une syntaxe différente (et oubliée par l'aide en ligne) : le premier argument est le numéro de canal d'entrée/sortie précédé du caractère dièse.

L'instruction `Close` ferme le fichier. Vous pouvez maintenant le relire dans un éditeur de textes.

Nous allons relire ligne par ligne ce fichier de texte avec la macro suivante.

```
rem Code08-03.sxw   bibli : LireEcrire Module2
Option Explicit

Sub LireFichierTexteVersion1()
Dim f1 As Integer, nomFichier As String
Dim uneLigne As String

f1 = FreeFile ' obtenir un numéro de fichier ouvert
Open nomFichier For Input As #f1
Do While not Eof(f1)
  Line Input #f1, uneLigne
  print uneLigne
Loop
Close #f1
End Sub
```

Nous ouvrons le fichier en lecture (anglais : *For Input*). Une erreur se déclenchera si ce fichier n'existe pas.

La fonction `Eof` renvoie `True` tant qu'il reste au moins une information à lire dans le fichier. Notez qu'ici, on ne met pas de dièse devant le numéro de référence.

L'instruction `Line Input` récupère une ligne complète dans la variable chaîne de caractères. Elle est utile pour lire le contenu exact d'un fichier texte quelconque. Nous affichons la ligne lue avec une instruction `print` dans sa syntaxe ordinaire.

Fichier texte pour sauver des données

Reprenons l'exemple du `Module1` en remplaçant les instructions `Print` par des instructions `Write`.

```
rem Code08-03.sxw   bibli : LireEcrire Module3
Option Explicit

Sub EcrireFichierTexteV2()
Dim f1 As Integer
Dim OuiNon As Boolean

OuiNon = True
f1 = FreeFile ' obtenir un numéro de fichier ouvert
Open nomFichier For Output As #f1
Write #f1, "Ceci est un texte."
Write #f1, ""                        ' ligne vide
Write #f1, "Troisième ligne du texte ", 12345, OuiNon
Close #f1
End Sub
```

Remarquez avec le deuxième `Write` que nous avons ajouté un argument en plus du numéro de canal, contrairement à la syntaxe indiquée par l'aide en ligne.

Relisez maintenant le fichier texte (avec un éditeur ou par la macro du `Module2`). Le contenu vous surprendra probablement :

```
"Ceci est un texte."
""
"Troisième ligne du texte ",12345,#True#
```

En fait, Basic écrit un texte de manière à pouvoir le relire en reconnaissant des éléments `String` ou des valeurs, séparés par des virgules. Ce n'est pas un texte ordinaire mais un *moyen de stocker des données*.

Supposons maintenant que nous connaissions la structure du fichier à lire, mais pas son contenu : nous savons seulement qu'il se compose d'un texte, puis un texte, puis un texte, puis un nombre, puis une valeur booléenne. Voici comment récupérer les valeurs correspondantes :

```
rem Code08-03.sxw   bibli : LireEcrire Module4
Option Explicit

Sub LireFichierTexteVersion2()
Dim f1 As Integer
Dim unTexte As String, unNombre As Long, unBool As Boolean

f1 = FreeFile ' obtenir un numéro de fichier ouvert
Open nomFichier For Input As #f1
Input #f1, unTexte
print unTexte
Input #f1, unTexte
print unTexte
Input #f1, unTexte, unNombre, unBool
print unTexte
print unNombre, unBool
Close #f1
End Sub
```

Les instructions `print` montrent que nous récupérons bien les informations originales, sans qu'elles soient modifiées par les caractères de délimitation qui se trouvent dans le fichier. Évidemment, si la structure du fichier diffère quelque peu de ce qu'on attend, les informations ne seront pas correctement récupérées et une erreur d'exécution pourra se produire.

Fichier binaire à accès direct

Le codage suivant va créer et utiliser un fichier binaire à accès direct (il ne s'agit plus d'un fichier texte). Dans un tel fichier, chaque enregistrement occupe un même nombre

d'octets. L'intérêt est de disposer de l'équivalent d'un tableau unidimensionnel comportant un très grand nombre d'éléments, et dans lequel l'accès à un élément quelconque ne nécessite pas d'avoir lu tous les éléments qui le précèdent.

```
rem Code08-03.sxw   bibli : LireEcrire Module5
Option Explicit

Sub EcrireRelireFichierAccesDirect()
Dim f1 As Integer
Const nomFichier = "C:\Docs OpenOffice\essai002.bin"
Dim uneValeur As Double, n As Long

f1 = FreeFile
' chaque enregistrement contiendra un Double = 8 octets
Open nomFichier For Random Access Write As #f1 Len=8
uneValeur = 2
for n = 1 to 100 ' écrire les puissances de 2 successives
  Put #f1,, uneValeur
  uneValeur = uneValeur * 2
next
Close #f1

' relecture du fichier
f1 = FreeFile
Open nomFichier For Random Access Read As #f1 Len=8
Get #f1, 10, uneValeur ' chercher 2 puissance 10
print uneValeur
Get #f1, 3, uneValeur ' chercher 2 puissance 3
print uneValeur
Get #f1, 32, uneValeur ' chercher 2 puissance 32
print uneValeur
Get #f1, 100, uneValeur ' chercher 2 puissance 100
print uneValeur
Close #f1
End Sub
```

L'instruction `Open` est utilisée avec le paramètre `For Random` qui indique un accès direct, suivi du paramètre `Access Write` précisant que seule l'écriture de fichier sera utilisée. Le paramètre `Len` indique la taille de chaque enregistrement ; ici, nous voulons stocker des nombres `Double`, qui occupent 8 octets chacun.

Dans la phase d'écriture, nous utilisons l'instruction `Put` pour écrire chaque enregistrement. La deuxième virgule sépare un paramètre inutilisé, le numéro d'enregistrement : les enregistrements sont écrits ici successivement.

Nous relisons ensuite le fichier en utilisant l'accès direct : le deuxième paramètre de l'instruction `Get` précise le numéro de l'enregistrement à rechercher. Si vous connaissez par cœur vos puissances de deux, vous pourrez vérifier que les valeurs obtenues sont correctes...

L'accès direct est aussi possible en écriture, mais seulement sur des enregistrements déjà écrits. Pour le démontrer, placez cette instruction juste avant la fermeture du fichier en écriture :

```
Put #f1,32, 111111111
```

La relecture vous renverra bien cette valeur pour l'enregistrement 32.

Fichier binaire pur

Le paramètre `For Binary` de l'instruction `Open` sert, en principe, à écrire des fichiers binaires de contenu quelconque. Cependant, des essais nous ont montré que, pour diverses raisons, cette fonctionnalité est inutilisable.

L'API OpenOffice.org offre le service `SimpleFileAccess`, qui permet lui aussi de manipuler des fichiers, et particulièrement les fichiers binaires. Nous aborderons ce sujet dans le chapitre 19.

Autres instructions Basic pour fichiers ouverts

Les fonctions et instructions suivantes sont utilisables sur des fichiers ouverts. Elles ne présentent pas de difficulté particulière, aussi nous vous renvoyons à la lecture de l'aide en ligne :

- `Reset` ferme tous les fichiers en cours ; elle est utile dans les traitements d'erreur.
- `Loc` et `Seek` concernent la position courante dans le fichier traité.
- `Eof` indique si la lecture est arrivée en fin de fichier.
- `Lof` renvoie la taille du fichier.
- `FileAttr` renvoie le mode d'ouverture du fichier, ou son *handle* (information qui ne concerne que les spécialistes).

Fonctions système

La fonction `Environ` permet de récupérer la valeur d'une variable d'environnement. Il n'existe malheureusement pas de moyen pour modifier une variable d'environnement par Basic.

L'instruction `Beep` sert à émettre un signal sonore. Sous MS-Windows, il s'agit du `Son par défaut`.

La fonction `GetSolarVersion` peut vous aider à déterminer la version exacte d'OpenOffice.org sur laquelle la macro s'exécute.

Les fonctions `TwipsPerPixelX` et `TwipsPerPixelY` servaient à convertir en pixels les dimensions d'éléments de boîtes de dialogue. Ce système de mesure n'est plus utilisé.

Lancer un programme externe

L'instruction `Shell` permet de lancer l'exécution d'un autre programme. Attention, sous Windows, il est préférable de convertir l'adresse au format URL, grâce à la fonction `ConvertToURL`, car les chemins comportant des espaces ou autres caractères non alphabétiques ne sont pas reconnus.

Les applications internes de Windows n'existent pas sous forme d'un fichier exécutable, donc `Shell` ne les trouvera pas. Le problème est contourné en écrivant un fichier batch `.bat` qui lance le programme, et en lançant le batch par `Shell`. Cet exemple de fichier batch écrit la version exacte de Windows dans un fichier texte.

```
Ver >version.txt
```

Un autre besoin est, par exemple, d'afficher un document HTML avec le navigateur assigné par défaut à ce type de fichier. Ceci n'est pas réalisable par `Shell`, mais nécessite une autre méthode qui utilise l'API d'OpenOffice.org. Nous la verrons au chapitre 19.

Conclusion

OOoBasic propose de nombreuses fonctionnalités intrinsèques. Du traitement des chaînes de caractères à la manipulation des fichiers, toutes ces instructions concourent à l'élaboration de programmes de plus en plus complets.

Le chapitre suivant expose comment traiter les erreurs apparaissant en cours d'exécution dans nos programmes parfois longs et complexes.

9

Le traitement des erreurs d'exécution

Nous allons traiter ici de la manière d'écrire des programmes Basic robustes, car capables de réagir raisonnablement face à un événement anormal. Le programme pourra ainsi afficher un message compréhensible pour l'utilisateur et fermer correctement les documents qu'il était en train de traiter, ou encore demander à l'utilisateur s'il souhaite continuer le traitement ou l'arrêter.

Nous verrons enfin que ce même principe peut être utilisé pour simplifier certains codages.

Le mécanisme d'interception d'erreur

Le principe de traitement d'une erreur d'exécution est de dérouter l'exécution vers une adresse particulière où se trouve un codage qui va analyser l'erreur et réagir de façon adéquate. On utilise pour cela une instruction très similaire au `GoTo`, dont nous avons dit grand mal au chapitre 6. Nous expliquerons ses multiples possibilités avec des exemples volontairement simples.

Un exemple typique

```
rem Code09-01.sxw   bibli : Erreurs Module1
Option Explicit

Sub TraitementErreurV1()
Dim chemFich As String

Do ' boucler jusqu'à ce que l'utilisateur réponde Annuler
  chemFich = InputBox("Chemin du fichier ?")
  if Len(chemFich) = 0 then Exit Do
  On Error GoTo errFich ' protéger les instructions suivantes
  MsgBox "Taille du fichier " & FileLen(chemFich)
  On Error GoTo 0 ' supprimer le traitement d'erreur
  MsgBox "OK"
Loop
Exit Sub ' terminaison d'exécution du sous-programme

errFich:
  MsgBox("Ce fichier n'existe pas", 16)
  ' reprendre l'exécution après l'instruction en faute
  Resume Next
End Sub
```

Cet exemple typique consiste à afficher la taille (en octets) d'un fichier indiqué par l'utilisateur. N'hésitez pas à l'exécuter en pas à pas dans l'EDI. Répondez une première fois avec un fichier existant : la boucle s'exécute en séquence et les deux instructions `On Error` semblent ne rien faire de particulier.

Donnez maintenant une réponse incorrecte. Au lieu d'exécuter l'instruction `MsgBox` de la boucle, c'est le `MsgBox` situé à la suite de l'étiquette `errFich` qui est exécuté. En effet, l'instruction précédente stipule qu'en cas d'erreur (toute erreur), le programme doit aller à cette étiquette. L'instruction `Resume Next` force l'exécution à reprendre à la suite de celle qui avait causé une erreur.

L'instruction `On Error GoTo 0` a pour effet de revenir au traitement d'erreur par défaut (Basic affiche un message et stoppe le programme). Si elle n'existait pas, le traitement d'erreur mis en place resterait en vigueur pour les instructions suivantes. Supposez qu'une autre erreur se produise, par exemple une division par zéro dans un calcul : l'exécution serait déroutée de nouveau à l'étiquette `errFich` et un message sans rapport avec l'erreur s'afficherait. Réduisez donc la portée de votre traitement d'erreur au codage concerné.

PIÈGE Instruction Resume

Si vous utilisez l'instruction `Resume` toute seule, sans argument, l'instruction en faute sera ré-exécutée. Cette faculté est parfois utile. Cependant, si vous n'avez pas prévu le traitement en conséquence, vous entrerez dans une boucle infernale.

Conséquence d'une instruction non exécutée

Comme nous l'avons constaté, l'instruction déclenchant une erreur n'est pas exécutée. Supposons que nous ayons utilisé une variable intermédiaire (de type `Long`) :

```
longFich = FileLen(chemFich)
MsgBox longFich
```

Dans ce cas, au retour du traitement d'erreur, la variable `longFich` n'aurait pas reçu de valeur ; ou plus exactement, elle garderait la valeur qu'elle possédait avant l'instruction, éventuellement provenant du tour de boucle précédent. Ceci peut induire des erreurs logicielles dans la suite du traitement. Vous devez penser à cet aspect dans votre traitement d'erreur. Le prochain exemple nous donnera une solution.

Reprise du traitement à un autre endroit

Reprenons l'exemple précédent en ajoutant une autre instruction qui affiche les attributs du fichier.

```
rem Code09-01.sxw   bibli : Erreurs Module2
Option Explicit

Sub TraitementErreurV2()
Dim chemFich As String, repertoire As String

Do ' boucler jusqu'à ce que l'utilisateur réponde Annuler
  chemFich = InputBox("Chemin du fichier ?")
  if Len(chemFich) = 0 then Exit Do
  On Error GoTo errFich ' protéger les instructions suivantes
  MsgBox "Taille du fichier " & FileLen(chemFich)
  ' - - autres instructions liées au fichier - -
  MsgBox "Attributs du fichier : " & GetAttr(chemFich)
 suite1:
  On Error GoTo 0 ' supprimer le traitement d'erreur
  ' - - autres instructions non liées au fichier - -
  MsgBox "OK"
Loop
Exit Sub ' terminaison d'exécution du sous-programme

errFich:
  MsgBox("Ce fichier n'existe pas", 16)
  ' éviter la poursuite du traitement normal
  Resume suite1' reprendre à l'étiquette suite1
End Sub
```

Si nous avions un traitement d'erreur `Resume Next`, l'affichage des attributs déclencherait une nouvelle erreur. Il est certain qu'en cas d'erreur sur la première instruction, les autres

utilisant `chemFich` ne signifient plus rien. L'instruction `Resume suite1` nous permet de reprendre à un endroit plus approprié de notre codage.

Nous avons ici une solution, mais aussi une source d'ennuis si nous effectuons des sauts un peu n'importe où dans le codage, car Basic exécutera aveuglément le saut, même si cela conduit à un bouclage.

Pour éviter les programmes « plats de spaghetti » suivez ces conseils :

- Limitez la taille de vos sous-programmes.
- Écrivez d'abord le fonctionnement sans erreur du sous-programme, terminé par un `Exit Sub` (ou `Exit Function`).
- Placez en fin de sous-programme les traitements d'erreurs.
- Faites des traitements d'erreur simples.
- Relisez attentivement votre programme en exécutant mentalement chaque cas d'erreur, y compris la reprise du cours normal d'exécution après son traitement.

L'instruction Resume est indispensable

Vous pouvez avoir l'idée de faire un traitement d'erreur se terminant par un simple `GoTo` pour continuer l'exécution, sans exécuter une instruction `Resume`. C'est une mauvaise idée, car l'instruction `Resume` sert aussi à désactiver l'erreur en cours. Sans elle, l'exécution de Basic stoppera à la prochaine erreur ou au prochain `On Error GoTo etiquette`.

Dans un traitement d'erreur, l'instruction `Resume` doit apparaître le plus rapidement possible pour éviter qu'une deuxième erreur ne se produise avant la fin du traitement.

Ignorer les erreurs

Il peut être nécessaire d'ignorer volontairement les erreurs d'exécution. C'est en particulier le cas, suite à une erreur fatale, dans la phase de libération des ressources avant d'arrêter le programme. Basic propose pour cela l'instruction :

```
On Error Resume Next
```

Cette variante d'instruction `On Error` ne doit pas être confondue avec l'instruction `Resume Next`. Elle est en fait équivalente à :

```
On Error GoTo etiquetteInvisible
' - - - -
' - - - - autres instructions
' - - - -
Exit Sub
etiquetteInvisible:
  Resume Next
```

L'exemple suivant déclenche volontairement deux erreurs, mais le programme s'achèvera malgré tout sans message d'erreur ainsi que le montrent les instructions `Print`.

```
rem Code09-01.sxw   bibli : Erreurs Module3
Option Explicit

Sub IgnorerErreurs()
Dim v1 As Double

On Error Resume Next
MsgBox FileLen("") ' première erreur ignorée
print "Repère 1"
v1 = 0
v1 = 37/v1        ' deuxième erreur ignorée
print "Repère 2"
On Error GoTo 0
End Sub
```

Évidemment, ce genre de « non-traitement » doit être réservé à des cas justifiés, et dans les séquences les plus simples possibles.

Informations sur l'erreur

Quelques fonctions Basic peuvent servir à analyser l'erreur survenue :

- `Erl` renvoie le numéro de la ligne ayant déclenché l'erreur.
- `Err` renvoie un numéro d'erreur.
- `Error` renvoie le texte explicatif de l'erreur, dans la langue de la version localisée d'OpenOffice.org.

La liste des numéros d'erreurs est bien cachée dans l'aide en ligne (page « Débogage d'un programme Basic ») et les textes explicatifs sont restés en anglais. Aussi nous la reproduisons dans le tableau 9-1 avec les textes de diagnostic français, parfois éloignés de la signification réelle. Nous avons ajouté d'autres codes non listés dans l'aide en ligne. Cette liste est susceptible d'augmenter avec les versions futures d'OpenOffice.org.

Tableau 9–1 Erreurs d'exécution Basic

Numéro	Diagnostic
1	Exception
2	Erreur de syntaxe
3	`Return` sans `Gosub`
4	Saisie incorrecte, veuillez recommencer
5	Appel de procédure incorrect

Tableau 9–1 Erreurs d'exécution Basic (suite)

Numéro	Diagnostic
6	Débordement
7	Mémoire insuffisante !
8	`Array` déjà dimensionné
9	Index en dehors de la plage définie
10	Définition faisant double emploi
11	Division par zéro
12	Variable indéfinie
13	Types de données incompatibles
14	Paramètre incorrect
18	Processus interrompu par l'utilisateur
20	Résumé sans erreur (signifie : instruction `Resume` employée sans erreur préalable)
28	Mémoire tampon insuffisante
35	La sous-procédure ou procédure fonctionnelle n'est pas définie
48	Erreur lors du chargement d'un fichier DLL
49	Convention d'appel DLL incorrecte
51	Erreur interne
52	Nom ou numéro de fichier incorrect
53	Fichier introuvable
54	Mode de fichier incorrect
55	Fichier déjà ouvert
57	Erreur de périphérique E/S
58	Un fichier de ce nom existe déjà
59	Longueur d'enregistrement incorrecte
61	Disquette/disque dur plein(e)
62	Lecture au-delà de la fin du fichier
63	Numéro d'enregistrement incorrect
67	Trop de fichiers
68	Périphérique indisponible
70	Accès refusé
71	La disquette n'est pas prête
73	Non implanté (signifie : fonction non implémentée)
74	Impossible de renommer sur des unités différentes !
75	Erreur d'accès au chemin/fichier

Tableau 9–1 Erreurs d'exécution Basic (suite)

Numéro	Diagnostic
76	Chemin introuvable
91	Variable d'objet non paramétrée
93	Échantillon de chaîne de caractères incorrect (signifie : chaîne de caractères non valide)
94	Utilisation du zéro interdite (signifie : utilisation incorrecte de `Null`)
250	Erreur DDE
280	Attente de réponse dans une connexion DDE
281	Aucun canal DDE libre
282	Aucune application ne réagit à la tentative de connexion DDE
283	Trop d'applications répondent à la tentative de connexion DDE
284	Canal DDE verrouillé
285	L'application externe ne peut pas exécuter l'opération DDE
286	Timeout pendant l'attente de la réponse DDE
287	L'utilisateur a appuyé sur ÉCHAP pendant l'opération DDE
288	L'application externe est occupée
289	Données non fournies dans l'opération DDE
290	Le format des données est incorrect
291	L'application externe a été quittée
292	La connexion DDE a été interrompue ou modifiée
293	Méthode DDE appelée sans avoir ouvert un canal DDE
294	Format de lien DDE incorrect
295	Le message DDE a été perdu
296	Le `paste link` a déjà été exécuté
297	Impossible de définir le `LinkMode` à cause d'un `Link-Topic` incorrect
298	Le DDE requiert `DDEML.DLL`
323	Impossible de charger le module à cause d'une erreur de format
341	Index des objets incorrects
366	L'objet n'est pas disponible (signifie : pas de document ou pas de vue active du document)
380	Valeur de propriété incorrecte
382	La propriété est en lecture seule
394	La propriété est en écriture seule
420	Référence d'objet incorrecte
423	Propriété ou méthode introuvable
424	Objet requis

Tableau 9–1 Erreurs d'exécution Basic (suite)

Numéro	Diagnostic
425	Utilisation incorrecte d'un objet
430	Cet objet ne supporte pas l'automatisation OLE
438	L'objet ne supporte pas cette propriété ou méthode
440	Erreur lors de l'automatisation OLE
445	L'objet indiqué ne supporte pas cette action
446	L'objet indiqué ne supporte pas les arguments cités
447	L'objet indiqué ne supporte pas l'environnement linguistique actuel
448	L'argument cité est introuvable
449	L'argument n'est pas facultatif
450	Nombre d'arguments incorrect
451	L'objet n'est pas une liste
452	Nombre ordinal non valable
453	La fonction DLL indiquée est introuvable
460	Format de presse-papiers incorrect
951	Symbole imprévu : xxxx
952	Requis : xxxx
953	Symbole requis
954	Variable requise
955	Étiquette requise
956	Impossible d'attribuer la valeur
957	La variable est déjà définie
958	La sous-procédure ou procédure fonctionnelle est déjà définie
959	L'étiquette est déjà définie
960	Variable introuvable
961	`Array` ou procédure introuvable
962	Procédure introuvable
963	L'étiquette n'est pas définie
964	Type de données inconnu
965	`Exit` requis
966	Bloc d'instructions encore ouvert : xxxx fait défaut
967	Erreur de parenthèses
968	Le symbole a déjà reçu une autre définition
969	Les paramètres ne correspondent pas à la procédure

Tableau 9–1 Erreurs d'exécution Basic (suite)

Numéro	Diagnostic
970	Le nombre contient un caractère incorrect
971	Vous devez dimensionner l'`Array`
972	`Else`/`Endif` sans `If`
973	xxxx interdit dans une procédure
974	xxxx interdit en dehors d'une procédure
975	Les dimensions indiquées ne concordent pas
976	Option inconnue : xxxx
977	La constante a été redéfinie
978	Le programme est trop volumineux
979	`Strings` ou `Arrays` inadmissibles
1000	L'objet ne possède pas cette propriété
1001	L'objet ne possède pas cette méthode
1002	L'argument requis fait défaut
1003	Nombre d'arguments incorrect
1004	Erreur dans l'exécution d'une méthode
1005	Impossible de définir la propriété
1006	Impossible de déterminer la propriété
65535	Fehler 65 535: Kein Fehlertext verfuegbar! (voir remarque)

REMARQUE **Humour germanique**

Si vous utilisez la fonction `Error` dans un programme sans erreur vous obtiendrez (version 1.1.1) le texte suivant :

Fehler 0: Kein FehlerText verfuegbar!

Ce qui signifie, en allemand, qu'il n'existe pas de texte pour l'erreur ayant le numéro zéro.

Le même message apparaît pour l'erreur 65535, qui n'est pas utilisée.

Portée d'un traitement d'erreur

Un traitement d'erreur en vigueur dans un sous-programme A sera utilisé par toute erreur déclenchée dans un sous-programme B appelé directement ou indirectement depuis le sous-programme A.

Dans cet exemple, nous allons déclencher et traiter une erreur dans le sous-programme `Etage1`, puis dans le sous-programme `Etage2`, lui-même appelé par `Etage1`. Les nombreux `print` repèrent l'avancement de l'exécution.

```
rem Code09-01.sxw   bibli : Erreurs Module4
Option Explicit

Sub PorteeTraitementErreurs
On Error goto TraiterErreur
print "Premier appel Etage1"
Etage1(0)   ' appel d'un sous-programme
print "Deuxième appel Etage1"
Etage1(1000)
print "Fin du programme principal"
Exit Sub

TraiterErreur:
  print "Erreur " & erl, error
  Resume Next
End Sub

Sub Etage1(v1 As Long)
Dim v2 As Long
print "Etage 1, argument " & v1
v2 = 1/v1
print "Appel Etage2"
Etage2(v1)   ' appel d'un sous-programme
End Sub

Sub Etage2(v6 As Integer)
Dim v7 As Integer
print "Etage 2, argument " & v6
v7 = v6 * v6
End Sub
```

À l'inverse, si un traitement d'erreur est mis en place dans un sous-programme appelé, il est désactivé au retour du sous-programme.

```
rem Code09-01.sxw   bibli : Erreurs Module5
Option Explicit

Sub ErreurNonTraitee()
Dim v3 As Integer

GererErreur   ' appel d'un sous-programme
v3 = 1000
v3 = 1000*v3 ' déclenche une erreur
End Sub
```

```
Sub GererErreur()
On Error goto TraiterErreur
print "Gestion d'erreur en place"
Exit Sub

TraiterErreur:
  print "Erreur " & erl, error
  Resume Next
End Sub
```

Le traitement d'erreurs pour simplifier le codage

Dans certains cas, le traitement d'erreurs peut s'avérer un moyen simple pour capturer différents cas anormaux qui méritent le même traitement. L'annexe B décrit la fonction utilitaire `getLocaleStyleName`, servant à traduire un nom de style, qui emploie justement cette méthode.

Déclencher une erreur

Dans le même esprit de simplification, il peut être intéressant de déclencher une erreur identique à un des codes d'erreur répertoriés dans le tableau 9-1. Par exemple, pour déclencher l'erreur 70, il suffit d'exécuter l'instruction suivante :

```
err = 70
```

Ce mécanisme a été utilisé pour lister tous les codes d'erreur, voir le document `Code09-02.sxw` dans le zip téléchargeable. Nous ne détaillerons pas la macro car elle nécessite des connaissances sur l'écriture dans Writer que nous verrons au chapitre 11.

Conclusion

La gestion des erreurs peut être considérée comme un travail ingrat mais se révèle absolument indispensable à la construction d'un programme robuste dans lequel l'utilisateur peut avoir confiance. Nous nous sommes attachés à présenter les méthodes permettant d'utiliser les fonctionnalités offertes par OOoBasic.

Ce chapitre clôt notre description de OOoBasic. Rappelons que l'aide en ligne est accessible et qu'il n'y a aucune honte à la consulter en cas de doute sur la syntaxe d'une instruction.

Fort de cette connaissance de OOoBasic, la partie suivante va nous plonger dans la manipulation des documents OpenOffice.org au travers de l'API.

Troisième partie

Manipuler les documents OpenOffice.org

Vous allez maintenant apprendre à écrire ou modifier des documents OpenOffice.org : Writer, Calc, Draw, etc. Vous aurez besoin d'utiliser l'API, mais nous évitons toute théorie en nous concentrant sur les solutions à des besoins réels. OpenOffice.org réutilise des concepts généraux dans chaque type de documents, mais avec des variations propres à chacun. Nous avons regroupé les principes communs dans le chapitre Accéder aux documents et les aspects spécifiques dans les chapitres suivants. Vous noterez cependant parfois des redondances apparentes, qui sont justifiées par des différences parfois subtiles. Les chapitres les plus importants sont évidemment ceux consacrés aux documents Writer, Calc et Draw. Ils sont assez indépendants, quoique nous vous renvoyions parfois à un autre chapitre (souvent celui consacré au module de traitement de texte Writer) où tel concept a été détaillé. Dans chacun de ces trois chapitres, il n'est nul besoin d'effectuer une lecture complète : après avoir acquis les notions de base, utilisez ensuite le livre comme une référence, et n'approfondissez que les sujets qui vous sont utiles.

Le chapitre concernant les objets insérés dans un document décrit des fonctions présentes dans différents types de documents. Ici l'approche consiste à montrer la méthode pour un premier type de document, puis l'équivalent sur les autres documents.

10

Les documents OpenOffice.org

À partir de ce chapitre, nous allons utiliser les ressources de l'API OpenOffice.org pour manipuler des documents avec des macros Basic. Ceci nous amènera à utiliser plusieurs concepts que nous n'avons pas vus jusqu'ici. En voici une description très succincte.

- **Les structures** de données sont des éléments qui regroupent plusieurs sous-données. On accède à chaque sous-donnée en ajoutant un point et le nom de celle-ci à droite du nom de la structure. Chaque sous-donnée est d'un type connu, ou elle-même une structure.
- **Les objets** sont des concepts logiciels qui regroupent plusieurs sous-programmes ou fonctions, et des données qui leur sont liées. Basic les considère comme des données du type `Object`. Notez cependant que le type `Variant` est aussi valable.
- **Les méthodes** d'un objet sont des sous-programmes ou fonctions utilisables seulement avec l'objet. On utilise une méthode en ajoutant un point et le nom de celle-ci à droite du nom de l'objet. Une fonction d'un objet peut fort bien renvoyer un objet, et ainsi de suite, car un objet peut contenir d'autres objets ou donner accès à un autre objet.
- **Les propriétés** d'un objet sont similaires à des variables de données, mais seulement utilisables avec l'objet ; chaque propriété possède un type. En général, une propriété peut être lue et écrite, mais parfois seulement lue ou seulement écrite. Basic permet de considérer comme une seule propriété deux méthodes simples servant l'une à écrire, l'autre à lire une même donnée interne. Par exemple, certains objets exposent une méthode `setText` pour écrire dans une donnée interne et une fonction `getText` pour

lire cette donnée. Basic simplifie l'écriture en permettant de lire ou écrire une pseudo-propriété `Text`. Notez que, dans le concept UNO, les propriétés sont liées à un service ; cependant, en pratique elles sont toujours liées à un objet.

Ne vous inquiétez pas trop de ces définitions, ce sera plus clair sur les utilisations pratiques.

POUR LES EXPERTS Programmation objet

L'API est orientée objet, mais OpenOffice.org Basic n'est pas un langage de programmation orienté objet. C'est pourquoi il a fallu réaliser certaines « contorsions » pour accéder à l'API depuis Basic. Voir l'annexe A.

Dans ce chapitre, nous allons décrire des aspects communs à tous les documents OpenOffice.org, en particulier comment charger un document, le sauvegarder, l'imprimer ; et comment convertir un document (les filtrages).

À RETENIR Noms des variables

Pour nos exemples de macros, nous utiliserons systématiquement pour les variables importantes des noms significatifs en français. Nous reprendrons les mêmes noms d'un exemple à l'autre, ce qui vous aidera à comprendre. Et pour nous, cela réduit l'effort de codage grâce à la plus belle invention de l'informatique : le copier/coller.

Accéder au document

Pour qu'une macro puisse manipuler un document OpenOffice.org, il est nécessaire de le référencer à travers une variable de type `Object`. Trois situations sont possibles :

- Nous voulons manipuler le document à partir duquel nous avons lancé la macro.
- Nous voulons manipuler un autre document, qui par exemple n'est pas encore ouvert.
- Nous voulons créer un nouveau document.

Chaque cas nécessite une méthode particulière, que nous allons expliquer. Une fois l'objet document obtenu, nous pourrons le manipuler (les principes dépendent du type de document, voir les chapitres suivants).

Accéder au document en cours

Ceci est le cas le plus courant. La macro est lancée à partir du document qui nous intéresse, soit par le menu Outils > Macros > Macro..., soit par un bouton d'une barre d'outils ou un bouton sur le document, soit par un raccourci clavier, etc.

Nous allons déclarer dans notre macro une variable objet et l'initialiser ainsi :

```
Dim monDocument As Object
monDocument = ThisComponent
```

Et voilà ! Nous avons maintenant une variable qui référence notre document. Le terme `ThisComponent` est une facilité offerte par Basic. Il a l'avantage de fonctionner même si la macro est exécutée dans l'EDI.

Il existe deux autres manières d'obtenir le document en cours, plus complexes et incompatibles avec une exécution dans l'EDI :

```
' méthode 2
Dim monDocument As Object
monDocument = StarDesktop.CurrentComponent

' méthode 3, strictement équivalente à la méthode 2
Dim monBureau As Object
Dim monDocument As Object
monBureau = createUnoService("com.sun.star.frame.Desktop")
monDocument = monBureau.CurrentComponent
```

Pour les raisons exposées plus haut, nous n'utiliserons que la première méthode. D'une manière générale, nous présenterons toujours les codages les plus simples.

> **À RETENIR StarDesktop**
>
> L'objet `StarDesktop` est l'application OpenOffice.org elle-même. Comme il est parfaitement possible de lancer une macro sans document ouvert, `StarDesktop` est alors le seul moyen d'obtenir certaines entités, comme la fenêtre courante, ou certaines routines.

Accéder à un autre document

Nous utiliserons pour cela l'objet `StarDesktop`, qui expose la méthode `LoadComponentFromURL`. Elle nous sert à récupérer un document à partir de son adresse fichier exprimée sous la forme d'une URL.

Les utilisateurs Unix sont familiers avec les adresses URL, mais pas les utilisateurs de MS-Windows. Voici une même adresse exprimée dans la syntaxe MS-Windows et dans la syntaxe URL :

```
C:\Docs OpenOffice\tata.sxc     ' adresse au format MS-Windows
file:///C|/Docs%20OpenOffice/tata.sxc ' adresse au format URL
```

Pour éviter des erreurs, il est préférable d'utiliser la fonction `convertToURL` de Basic, qui reçoit en argument une adresse dans le format du système d'exploitation et renvoie l'équi-

valent en syntaxe URL. Les deux adresses sont des chaînes de caractères. La fonction `convertFromURL` effectue la conversion inverse.

Le codage suivant réalise le chargement d'un document. Nous l'expliquerons juste après.

```
Dim monDocument As Object
Dim adresseDoc As String
Dim propFich()

adresseDoc = convertToURL("C:\Mes Documents\tata.sxc")

monDocument = StarDesktop.LoadComponentFromURL(_
              adresseDoc, "_blank", 0, propFich)
```

Le caractère _ à la fin de l'avant-dernière ligne sert uniquement à indiquer le prolongement de l'instruction sur la ligne suivante. Nous l'utilisons pour des raisons de mise en page, mais dans vos macros vous pouvez mettre toute l'instruction sur la même ligne.

La méthode `LoadComponentFromURL` comporte quatre arguments obligatoires. Les arguments employés ici correspondent au cas de loin le plus courant.

La variable `adresseDoc` est initialisée avec l'URL du document à charger. Elle sera transmise en premier argument à la méthode `LoadComponentFromURL` de l'objet `StarDesktop`.

Le deuxième argument est une chaîne de caractères, que vous devez écrire exactement telle qu'indiquée.

Le troisième argument est généralement la valeur zéro.

Le quatrième argument précise certaines options concernant le fichier. La variable `propFich` est un peu particulière : ici, elle sert seulement à indiquer que le quatrième argument est un tableau vide, c'est-à-dire que nous n'utilisons que les options par défaut. Un tableau vide se définit comme indiqué sur la ligne `Dim` correspondante : avec deux parenthèses sans indication de dimension. Le type de la variable n'est pas important dans ce cas, aussi nous utilisons le type par défaut, `Variant`.

POUR LES EXPERTS Autres arguments

Seuls des cas très particuliers peuvent nécessiter d'autres valeurs pour les arguments 2 et 3 de la méthode `LoadComponentFromURL`. Les experts informatiques trouveront de plus amples informations (en anglais) dans le chapitre 6.1.5 du Developer'sGuide, qui fait partie du SDK.

Si le document recherché n'existe pas, la variable `monDocument` recevra la valeur `Null`. Nous pouvons vérifier ceci :

```
if IsNull(monDocument) then
  MsgBox("Le document n'existe pas", 16)
end if
```

Si le document est déjà chargé, une deuxième fenêtre sera ouverte en lecture seule pour l'utilisateur. Vous pouvez ainsi afficher deux (ou plus) zones différentes du même document ; mais attention, car vous obtenez ainsi plusieurs références sur le même document.

Propriétés d'ouverture de document

Le quatrième argument de `LoadComponentFromURL` permet aussi de choisir une ou plusieurs option(s) de chargement. Chaque option est une structure appelée valeur de propriété (anglais : `PropertyValue`). Cette structure, très courante dans l'API, est composée de deux composantes :

- `Name`, de type `String`, contient le nom de la propriété ; respectez la casse (majuscules, minuscules) dans l'écriture de ce nom.
- `Value`, dont le type exact dépend de la propriété, contient la valeur désirée.

Pour transmettre plusieurs options dans un seul argument, nous devons déclarer un tableau dont chaque élément est une structure `PropertyValue`. Par exemple, la ligne ci-dessous déclare un tableau de deux structures, un élément d'index 0 et un élément d'index 1 :

```
Dim propFich(1) As New com.sun.star.beans.PropertyValue
```

Cette déclaration `Dim` utilise la forme `As New` qui précise que le type demandé est indiqué par la séquence qui suit. Cette séquence doit être écrite exactement, en respectant la casse. Chaque structure particulière utilisée par l'API possède une séquence spécifique, commençant toujours par `com.sun.star`.

En reprenant l'exemple précédent, nous allons supposer que le document est protégé par un mot de passe. Le mot de passe obtenu de l'utilisateur est transmis par l'option `Password`. La valeur du mot de passe est une chaîne de caractères.

```
Dim monDocument As Object
Dim adresseDoc As String
Dim propFich(0) As New com.sun.star.beans.PropertyValue

adresseDoc = convertToURL("C:\Docs OpenOffice\tata.sxc")

propFich(0).Name = "Password"
propFich(0).Value = "Julie37" ' le mot de passe
monDocument = StarDesktop.LoadComponentFromURL(_
              AdresseDoc,"_blank",0, propFich())
```

Cet exemple vous montre comment remplir un élément d'une structure `PropertyValue`. Remarquez enfin que le quatrième argument de `LoadComponentFromURL` utilise des parenthèses vides pour signifier à Basic que l'ensemble du tableau doit être transmis.

Les principales propriétés (options) utiles pour l'ouverture d'un document sont listées au tableau 10-1.

Tableau 10–1 Options pour l'ouverture d'un document

Propriété	Type	Signification
`Hidden`	`Boolean`	`True` pour charger le document sans le rendre visible ; valeur `False` par défaut.
`Password`	`String`	Mot de passe (en clair) pour charger un document OpenOffice.org protégé.
`ReadOnly`	`Boolean`	`True` pour ouvrir le document en lecture seule (pour l'usager, pas pour une macro) ; valeur `False` par défaut.
`Version`	`Integer`	Numéro de la version à charger (utilisation du système de versions)
`MacroExecutionMode`	`Integer`	Précise les conditions d'exécution des macros du document ; constante nommée.
`AsTemplate`	`Boolean`	Utilisation d'un modèle de document ; voir texte.
`FilterName`	`String`	Nom du filtre d'importation ; voir la section sur les filtres d'import/export.
`FilterOptions`	`String`	Options du filtre.

Le menu Outils>Options>OpenOffice.org>Sécurité précise les conditions habituelles d'exécution des macros, voir le chapitre 2. L'option `MacroExecutionMode` permet de tenir éventuellement compte de cette configuration. Les valeurs possibles de `MacroExecutionMode` sont des constantes listées au tableau 10-2. Certaines valeurs ont été introduites avec la version 2.0 d'OpenOffice.org, pour d'autres le texte explicatif s'en est trouvé modifié. Ces constantes sont de la forme :

```
com.sun.star.document.MacroExecMode.ALWAYS_EXECUTE_NO_WARN
```

Attention à la casse ! Les constantes nommées doivent être écrites en respectant les majuscules et minuscules, sinon elles ne seront pas reconnues.

Tableau 10–2 Constantes de MacroExecutionMode

Constante	Signification
`NEVER_EXECUTE`	Ne pas exécuter les macros. Valeur par défaut.
`FROM_LIST`	Exécuter les macros provenant d'un répertoire de fichiers de confiance. En dehors des répertoires de confiance, une confirmation sera demandée.
`ALWAYS_EXECUTE`	V1.1 : Toujours exécuter les macros, éventuellement avec un message d'avertissement. V2.0 : les macros dont le certificat est reconnu, ou provenant d'un répertoire de confiance, sont exécutées sans avertissement ; une confirmation est demandée si les macros ne sont pas dans un répertoire de confiance, et pas certifiées.

Tableau 10–2 Constantes de MacroExecutionMode

Constante	Signification
USE_CONFIG	Utiliser la configuration, ouvrir une fenêtre de dialogue si une confirmation est nécessaire.
ALWAYS_EXECUTE_NO_WARN	Exécuter les macros sans demande de confirmation ni avertissement.
USE_CONFIG_REJECT_CONFIRMATION	Utiliser la configuration, ne pas exécuter les macros si une confirmation est nécessaire.
USE_CONFIG_APPROVE_CONFIRMATION	Utiliser la configuration, exécuter les macros comme si la confirmation était acceptée.
FROM_LIST_NO_WARN	V2.0 : seules les macros provenant d'un répertoire de fichiers de confiance seront exécutées.
FROM_LIST_AND_SIGNED_WARN	V2.0 : les macros dont le certificat est reconnu, ou provenant d'un répertoire de confiance, sont exécutées sans avertissement ; une confirmation est demandée si les macros ont un certificat inconnu ; si les macros ne sont ni dans un répertoire de confiance, ni certifiées, elles ne sont pas exécutées.
FROM_LIST_AND_SIGNED_NO_WARN	V2.0 : seules les macros dont le certificat est reconnu, ou provenant d'un répertoire de confiance, sont exécutées. Aucun avertissement n'est signalé.

Demander le mot de passe du document

Nous venons de voir que pour charger un document protégé par un mot de passe, il faut utiliser l'option `Password` dans la méthode `LoadComponentFromURL`.

OpenOffice.org offre un service permettant de demander le mot de passe à l'utilisateur de manière standardisée.

```
Dim monDocument As Object, demandePasse As Object
Dim adresseDoc As String
Dim propFich(0) As New com.sun.star.beans.PropertyValue

adresseDoc = convertToURL("C:\Docs OpenOffice\tata.sxc")
' utiliser le dialogue de demande de mot de passe
demandePasse = CreateUnoService( _
                    "com.sun.star.task.InteractionHandler")
propFich(0).Name = "InteractionHandler"
propFich(0).Value = demandePasse
monDocument = StarDesktop.LoadComponentFromURL(_
              AdresseDoc,"_blank",0, propFich())
```

La fonction Basic `CreateUnoService` renvoie un objet permettant d'utiliser le service `InteractionHandler`. Ensuite, au lieu d'utiliser l'option `Password`, on transmet cet objet dans l'option `InteractionHandler`. Respectez la casse dans le nom complet du service.

Une boîte de dialogue va s'ouvrir ; si l'utilisateur répond avec un mot de passe incorrect, un message d'erreur sera affiché par OpenOffice.org et la variable `monDocument` recevra la valeur `Null` ; si l'utilisateur annule le dialogue, il n'y aura pas de message et la variable `monDocument` recevra aussi la valeur `Null`.

Éditer un document modèle

Les documents « modèle » (anglais : Template) ont une extension spéciale, `stw` pour un modèle Writer, `stc` pour un modèle Calc, etc.

Charger un modèle pour le modifier requiert une option particulière, `AsTemplate`. Cette option, de type `Boolean`, doit prendre la valeur `False` afin de charger effectivement le modèle. Sinon, on crée un nouveau document reprenant ce modèle.

Créer un nouveau document

Pour créer un nouveau document OpenOffice.org, on utilise la même méthode que pour charger un document existant. Cependant, ici, l'adresse du document est une convention pour indiquer le type de document à créer. Par exemple, pour un document Calc, nous utiliserons :

```
adresseDoc = "private:factory/scalc"
```

Le tableau 10-3 liste les différentes pseudo-adresses possibles. L'adresse doit être reproduite exactement dans votre instruction. Le nouveau document obtenu suit le modèle par défaut en vigueur pour votre configuration d'OpenOffice.org.

Tableau 10–3 Pseudo-adresses pour créer un document

Type de document	Pseudo-adresse
Texte Writer	`private:factory/swriter`
Tableur Calc	`private:factory/scalc`
Dessin Draw	`private:factory/sdraw`
Présentation Impress	`private:factory/simpress`
Éditeur de formules Math	`private:factory/smath`

Créer un nouveau document conforme à un modèle

Créer un document qui soit la copie d'un document servant de modèle est une méthode qui peut simplifier considérablement vos macros. En effet, vous pouvez créer manuellement le document modèle de façon à ce qu'il comporte tous les styles de page, de paragraphe, de caractère, des en-têtes et bas de page, des pages en portrait et des pages en paysage, des images intégrées, des sections, des tableaux, un texte initial et des points de

repérage réalisés avec des signets. Ensuite, vous n'aurez plus qu'à compléter le document avec votre macro au lieu de tout créer *ab nihilo*. Ce principe est aussi valable pour un document Calc, Draw, ou autre.

Le chapitre 19 vous indique comment retrouver le(s) répertoire(s) des modèles dans votre installation OpenOffice.org.

Il existe deux méthodes pour créer un nouveau document à partir d'un modèle. La plus simple est de partir d'un vrai document modèle, c'est-à-dire avec une extension `stw` pour Writer, `stc` pour Calc, etc. Il suffit de faire comme si on chargeait le document modèle :

```
adresseDoc = convertToURL("C:\Mes modeles\Doc2colonnes.stw")
```

Ce document ne sera pas chargé, mais un nouveau document ordinaire sera créé, dont le contenu sera identique. Dans l'exemple, nous obtiendrons un nouveau document Writer.

La deuxième méthode consiste à prendre comme modèle un document ordinaire. Le document sera chargé avec la méthode déjà vue, mais en utilisant l'option `AsTemplate`. En affectant à cette option la valeur booléenne `True`, le document référencé sera pris comme modèle pour créer un nouveau document. Attention : vous devez utiliser comme référence un document ordinaire (Writer, Calc, etc), mais pas un « modèle ».

> **À RETENIR monDocument**
>
> Nous utiliserons systématiquement dans les exemples de l'ouvrage le document courant. Cependant, l'important est en fait la variable `monDocument`. Qu'elle soit initialisée avec le document courant ou avec un autre document ne change rien au reste du codage.

Sauver un document

Avant de sauver...

Si votre document a été modifié, par macro ou par l'utilisateur, il est nécessaire de le sauvegarder avant de le fermer. La propriété `isModified` de l'objet document est du type `Boolean`. Elle vaut `True` si le document a été modifié.

```
if monDocument.isModified then
  ' mettre ici les instructions pour sauver le document
end if
```

Cette même propriété peut être modifiée par programme, par exemple pour forcer l'apparition du dialogue de sauvegarde si l'utilisateur ferme le fichier :

```
monDocument.isModified = true
```

Le document expose la propriété `hasLocation`, de type `Boolean`, qui vaut `True` s'il possède une adresse, ou `False` si le document est nouveau, pas encore sauvegardé. La propriété `Location`, de type `String`, vous indique l'URL du fichier original.

```
if monDocument.hasLocation then
  adresseDoc = monDocument.Location
else
  adresseDoc = convertToURL("C:\Mes Documents\toto.sxc")
end if
```

Réaliser la sauvegarde

Sauver un document existant se fait très simplement, grâce à la méthode `Store` de l'objet document :

```
monDocument.Store
```

Si vous avez créé un nouveau document jamais encore sauvé, vous devez préciser le nom et l'adresse du fichier à créer, avec une URL. La méthode `storeAsURL` de l'objet document effectue alors la sauvegarde. Elle est l'équivalent de la commande de menu Fichier > Enregistrer sous... En reprenant les mêmes variables que dans les exemples précédents, on écrira dans le cas le plus simple :

```
Dim propFich2()

adresseDoc = convertToURL("C:\Docs OpenOffice\toto.sxc")
monDocument.storeAsURL(adresseDoc, propFich2)
```

Si le répertoire indiqué n'existe pas, il sera créé automatiquement.

Ici aussi, nous avons la possibilité d'utiliser des options de sauvegarde. Notre exemple n'utilise aucune option particulière. Les options principales utiles dans une opération de sauvegarde sont listées au tableau 10-4. Le principe d'initialisation des options est identique à ce que nous avons vu pour les options de chargement d'un fichier.

Notez que si vous utilisez l'option `Password`, toute valeur de mot de passe est utilisable, y compris une chaîne de caractères de longueur nulle.

Tableau 10–4 Options de sauvegarde d'un document

Propriété	Type	Signification
`Overwrite`	`Boolean`	`True` pour écraser un document déjà existant (valeur par défaut) `False` pour ne pas écraser le document (une exception sera déclenchée en cas de tentative de sauvegarde).
`Password`	`String`	Mot de passe pour chiffrer le document.
`Author`	`String`	Auteur de la version du document (utilisation du système de versions).
`Version`	`Integer`	Numéro de version à sauvegarder (utilisation du système de versions).
`Comment`	`String`	Commentaire sur la version (utilisation du système de versions).
`FilterName`	`String`	Nom du filtre d'exportation ; voir la section sur les filtres d'import/export.
`FilterOptions`	`String`	Options du filtre.

Enregistrer une copie

Si vous avez ouvert un fichier `tata.sxc` et l'avez sauvegardé par la méthode `storeAsURL` sous le nom `toto.sxc`, vous travaillez maintenant dans le document `toto.sxc`.

En revanche, la méthode `storeToURL` réalise une copie du document en cours, et seulement cela. Ainsi, travaillant sur `tata.sxc`, vous faites une copie appelée `tata001.sxc` et continuez sur `tata.sxc`. Ceci est utile pour des sauvegardes régulières.

La méthode `storeToURL` utilise la même syntaxe que `storeAsURL`.

Fermer le document

Divers auteurs, dont nous-mêmes, utilisaient auparavant une seule instruction pour fermer le document :

```
monDocument.dispose
```

Cette simplicité est révolue. Pour traiter des cas de configuration assez complexes à expliquer, et pour assurer la compatibilité avec les évolutions futures, la méthode recommandée pour fermer un document (ou une fenêtre) est la suivante :

```
On Error Resume Next ' ignorer l'erreur éventuelle
monDocument.close(True)
On Error GoTo 0 ' reprendre le traitement d'erreur normal
```

La méthode `close` signale à d'autres programmes utilisateurs que ce document va être fermé. Un de ces programmes peut manifester son mécontentement en déclenchant une erreur, que nous ignorerons. L'argument `True` dans la méthode `close` signale à ces autres programmes que, s'ils souhaitent continuer quand même, ils doivent en assumer la responsabilité (en ce qui concerne la libération des ressources). Vous pourrez lire dans le

« Developer's Guide », chapitre 6.1.5 paragraphe « Closing document », une discussion très technique de 7 pages sur ce sujet.

Exemples récapitulatifs

Nous allons présenter maintenant des exemples complets de macros qui utilisent les notions décrites jusqu'ici dans ce chapitre. Ces macros sont disponibles dans le zip téléchargeable. Dans le premier exemple, nous exécutons une macro sur le document en cours. Dans ce cas particulier, la macro est incluse dans le document lui-même.

```
rem Code10-04.sxw   bibli : LeDocument Module1
Option Explicit

Sub DocumentEnCours()
Dim monDocument As Object
monDocument = ThisComponent

' afficher tout le texte du document
MsgBox(monDocument.Text.String)
if MsgBox("Fermer ce document ?",4) = 6 then
  ' réponse = Oui
  on Error Resume Next ' ignorer l'erreur éventuelle
  monDocument.close(True)
  On Error GoTo 0 ' reprendre le traitement d'erreur normal
end if
End Sub
```

L'instruction permettant d'afficher le texte du document Writer sera expliquée dans le chapitre 11. Nous l'employons simplement pour prouver que nous accédons bien au document. L'instruction `MsgBox` utilise le paramétrage pour afficher un bouton Oui et un bouton Non. Relisez éventuellement la description de cette instruction au chapitre 8.

Dans le deuxième exemple, nous créons un nouveau document Calc à partir d'une autre macro du document Writer. Ce document est modifié, puis sauvé à une adresse particulière, en lui mettant un mot de passe. Changez éventuellement cette adresse pour être compatible avec votre système. Ce nouveau document est finalement fermé. Les explications suivent le codage.

```
rem Code10-04.sxw   bibli : LeDocument Module2
Option Explicit

' modifiez éventuellement cette ligne selon votre système
Public Const leDoc = "C:\Docs OpenOffice\DocHeure.sxc"
```

```
Sub CreerDocumentAvecPasse()
Dim monDocument As Object
Dim adresseDoc As String
Dim propFich()

adresseDoc ="private:factory/scalc" ' nouveau document Calc
monDocument = StarDesktop.LoadComponentFromURL(_
              adresseDoc, "_blank", 0, propFich)
MsgBox "Un nouveau document Calc doit être affiché maintenant"

' écrire l'heure dans la cellule A1 de la première feuille
Dim maCellule As Object
maCellule = monDocument.Sheets(0).getCellRangeByName("A1")
maCellule.String = "Ecrit le = " & Date & " Heure " & Time
MsgBox "Tableur modifié"

' convertToURL est inutile avec un système Linux
adresseDoc = convertToURL(leDoc)
' sauvegarder le document avec un mot de passe
Dim propFich2(0) As New com.sun.star.beans.PropertyValue
propFich2(0).Name = "Password"
propFich2(0).Value = "OpenOffice" ' ceci est le mot de passe !
monDocument.storeAsURL(adresseDoc, propFich2())

MsgBox "Fermeture du document"
on Error Resume Next
monDocument.close(True)
On Error GoTo 0
End Sub
```

La constante `leDoc` est déclarée `Public` car nous l'utiliserons aussi dans un autre module de la bibliothèque de macros. Après avoir ouvert un nouveau document Calc, nous écrivons un texte (la date et heure actuelles) dans la cellule A1 de la première feuille. Ne vous préoccupez pas de la méthode employée, le but est seulement de montrer que nous pouvons modifier le document. La variable `adresseDoc` est initialisée avec l'URL de sauvegarde du document. Pour définir l'option « mot de passe », nous utilisons une autre variable, car nous avons besoin maintenant d'un tableau à un élément de structure `PropertyValue`.

SÉCURITÉ Mots de passe

Pour simplifier l'exemple, le mot de passe apparaît en clair dans votre macro. Il peut être judicieux de la protéger elle-même par un mot de passe comme vu au chapitre 3 pour en éviter une lecture non autorisée.

Dans le troisième exemple, nous allons charger de nouveau le document Calc créé précédemment, le modifier puis le refermer.

```
rem Code10-04.sxw   bibli : LeDocument Module3
Option Explicit

Sub ChargerDocumentAvecPasse()
Dim monDocument As Object
Dim adresseDoc As String
Dim propFich(0) As New com.sun.star.beans.PropertyValue

' leDoc est déclaré dans le module 2
adresseDoc = convertToURL(leDoc)
propFich(0).Name = "Password"
propFich(0).Value = "OpenOffice" ' ceci est le mot de passe !
monDocument = StarDesktop.LoadComponentFromURL(_
              AdresseDoc,"_blank",0, propFich())

if IsNull(monDocument) then
  MsgBox("Le document n'existe pas" & chr(13) & _
    "ou le mot de passe est incorrect", 16)
  Stop ' arrêter l'exécution de Basic
end if
MsgBox "Tableur chargé"

' écrire l'heure dans la cellule A2 de la première feuille
Dim maCellule As Object
maCellule = monDocument.Sheets(0).getCellRangeByName("A2")
maCellule.String = "Ecrit le = " & Date & " Heure " & Time
MsgBox "Tableur modifié"

monDocument.store   ' sauvegarder (avec le mot de passe)
on Error Resume Next
monDocument.close(True) ' fermer le document
On Error GoTo 0
End Sub
```

Nous réutilisons la variable `leDoc` du module précédent afin d'obtenir l'URL du document à charger. Ici aussi, le mot de passe est dans le codage. Nous vérifions que le document a bien été chargé en testant le contenu de la variable `monDocument`. Si ce n'est pas le cas, on s'arrête là (l'instruction `chr(13)` sert à changer de ligne dans le message). Dans le cas normal, on écrit un texte dans la cellule A2, on sauvegarde le document et on le ferme.

L'objet `StarDesktop` est un conteneur de tous les documents ouverts d'OpenOffice.org. La macro suivante illustre l'accès tour à tour à tous les documents à partir de celui-ci en le fermant (sauvegardez vos documents avant d'exécuter la macro). N'ouvrez pas l'EDI pour lancer cette macro, mais utilisez plutôt le menu Outils > Macros > Macro.

```
rem Code10-04.sxw   bibli : LeDocument Module5

Sub BoucleFermeDocuments()
'pensez à sauvegarder vos documents
dim lesDocs, laCollection, leDoc

on error resume next

lesDocs=StarDesktop.components
laCollection=lesDocs.createEnumeration

while laCollection.hasMoreElements
   leDoc=laCollection.nextElement
   print "Fermeture du document " & leDoc.URL
   leDoc.close(True)
wend
End Sub
```

La collection `Components` est accessible depuis `StarDesktop` et expose la méthode `createEnumeration` afin de créer la collection des documents ouverts. La méthode de parcours est basée sur les deux propriétés `hasMoreElements` (reste-t-il des éléments à explorer ?) et `nextElement` (accéder à l'élément suivant que nous mettons dans la variable `leDoc`). Pour chaque élément, nous affichons son URL et envoyons une commande de fermeture.

Cette méthode peut être utile pour fermer OpenOffice.org (ce qui revient à fermer tous les documents ouverts) à deux exceptions près :

- Si l'EDI est ouverte - Le composant contenant l'EDI ne se ferme pas comme un document classique, mais en utilisant `leDoc.dispose()`. Ceci justifie la présence de la ligne `on error resume next` permettant de poursuivre le traitement en cas d'erreur.
- Sous Windows, si le lanceur rapide est en cours d'exécution - Seules les fenêtres d'OpenOffice.org seront fermées et le programme restera en cours d'exécution.

À partir de ces exemples vous pouvez réaliser des variantes mettant en oeuvre d'autres fonctionnalités parmi celles décrites. Signalons que l'exemple concernant les filtres d'import/export charge le même document en demandant le mot de passe à l'utilisateur.

Les filtres d'import/export

Les filtres d'importation servent à convertir un document d'un format autre que ceux d'OpenOffice.org vers un des formats d'OpenOffice.org. Les filtres d'exportation permettent d'enregistrer un document OpenOffice.org dans un autre format. Avant de vous lancer dans un projet Basic pour convertir des séries de documents, consultez le site OOo Converter (http://oooconv.free.fr/) géré par un des auteurs de ce livre. Il vous propose un tel outil, disponible en téléchargement ou utilisable directement en ligne.

Principes de base

La conversion revient à charger ou sauvegarder un document en utilisant une option particulière : `FilterName`. Cette option reçoit une chaîne de caractères spécifique au format concurrent. La liste complète des filtres reconnus par la version OpenOffice.org 1.1.1 se trouve dans le fichier `ListeDesFiltres.sxc` disponible dans le zip téléchargeable. Le nom du filtre choisi doit être recopié exactement (sous forme de chaîne de caractères) dans la valeur de l'option `FilterName`. Certains filtres peuvent nécessiter des paramètres, qui sont transmis dans l'option `FilterOptions`, de type `String`. Malheureusement, ces paramètres ne sont pas actuellement documentés, à l'exception du filtre CSV.

Pour charger un document d'un format concurrent, l'option `FilterName` est en général inutile car OpenOffice.org est assez intelligent pour trouver le filtre adéquat. Une fois le document chargé, il est considéré comme un document OpenOffice.org. Une sauvegarde `storeAsURL` ou `storeToURL` sans l'option `FilterName` produira un fichier au format par défaut défini par le menu Outils > Options > OpenOffice.org > Chargement/Enregistrement > Général.

En revanche, pour sauvegarder le document sous un format concurrent, il est nécessaire de préciser ce format avec l'option `FilterName`. Sous MS-Windows, il est souhaitable de donner une extension de fichier correspondant au format demandé, bien que ceci n'ait aucune influence sur le format obtenu.

Le tableau 10-5 liste les principaux formats pour importer ou exporter des documents en formats concurrents. Notez que les filtres pour MS-Office 97 permettent d'exporter des documents entièrement compatibles avec les versions ultérieures 2000 et XP.

Tableau 10–5 Valeurs de FilterName

FilterName	Commentaire
`dBase`	Conversion d'une table dBase en document Calc. Le jeu de caractères à utiliser doit être précisé dans l'option `FilterOptions`.
`DIF`	Data Interchange Format.
`HTML (StarCalc)`	Importer un fichier HTML et obtenir un document Calc.
`HTML (StarWriter)`	Importer un fichier HTML et obtenir un document Writer.
`Lotus`	
`Lotus 1-2-3 1.0 (DOS) (StarWriter)`	
`Lotus 1-2-3 1.0 (WIN) (StarWriter)`	
`MS Excel 4.0`	
`MS Excel 4.0 Vorlage/Template`	Pour un modèle de document.

Tableau 10–5 Valeurs de FilterName (suite)

FilterName	Commentaire
MS Excel 5.0/95	
MS Excel 5.0/95 Vorlage/Template	Pour un modèle de document.
MS Excel 95	
MS Excel 95 Vorlage/Template	Pour un modèle de document.
MS Excel 97	
MS Excel 97 Vorlage/Template	Pour un modèle de document.
MS PowerPoint 97	
MS PowerPoint 97 Vorlage	Pour un modèle de document.
MS WinWord 5	
MS WinWord 6.0	
MS Word 95	
MS Word 95 Vorlage	Pour un modèle de document.
MS Word 97	
MS Word 97 Vorlage	Pour un modèle de document.
Rich Text Format	
SYLK	
Text	Exporter en texte pur avec l'encodage du système d'exploitation.
Text - txt - csv (StarCalc)	
Text (encoded)	Import/export de texte pur encodé.

Exporter en PDF, Flash, HTML

Le tableau 10-6 liste les filtres d'exportation aux formats Adobe PDF, Macromedia Flash et HTML. Pour ces filtres, il est nécessaire d'utiliser la méthode `storeToURL`.

Pour un tableur, l'exportation au format Adobe PDF est restreinte aux zones d'impression éventuelles. Pour obtenir toutes les feuilles (non vides), il faut supprimer les zones d'impression. Cet exemple reprend le document Calc des exemples précédents et en crée une version PDF. Seule la fin de la macro concerne l'exportation.

```
rem Code10-04.sxw   bibli : LeDocument Module4
Option Explicit

Sub ConvertirEnPDF_DocumentAvecDemandePasse()
Dim monDocument As Object
Dim adresseDoc As String
Dim propFich(0) As New com.sun.star.beans.PropertyValue

' leDoc est déclaré dans le module 2
adresseDoc = convertToURL(leDoc)
' utiliser le dialogue de demande de mot de passe
demandePasse = CreateUnoService("com.sun.star.task.InteractionHandler")
propFich(0).Name = "InteractionHandler"
propFich(0).Value = demandePasse
monDocument = StarDesktop.LoadComponentFromURL(AdresseDoc,"_blank",0, propFich())

if IsNull(monDocument) then
  MsgBox("Le document n'existe pas" & chr(13) & "ou le mot de passe est incorrect", 16)
  Stop ' arrêter l'exécution de Basic
end if

propFich(0).Name = "FilterName"
propFich(0).Value = "calc_pdf_Export"
adresseDoc = convertToURL("C:\Docs OpenOffice\resultat.pdf")
monDocument.storeToURL(adresseDoc, propFich())
on Error Resume Next
monDocument.close(True)
On Error GoTo 0
End Sub
```

Tableau 10–6 Exportation en PDF, Flash, HTML

FilterName
calc_pdf_Export
draw_pdf_Export
impress_pdf_Export
math_pdf_Export
writer_pdf_Export
writer_web_pdf_Export
draw_flash_Export
impress_flash_Export
HTML
draw_html_Export
impress_html_Export

L'export en PDF peut comporter des paramètres supplémentaires. Ils sont transmis par une propriété `FilterData` dont la valeur est un tableau (array) de propriétés. Ce dernier peut comporter les propriétés suivantes :

- `CompressMode` pouvant prendre pour valeur :
  0 optimisé pour l'écran
  1 optimisé pour l'impression
  2 optimisé pour la presse à imprimer ;
- `PageRange` contenant une chaîne de caractères listant les pages à exporter, comme pour une impression manuelle.

Voici un exemple utilisant ces paramètres :

```
Sub PDFpartielWriter()
Dim monDocument As Object
Dim adresseDoc As String
Dim PropFich(1) As New com.sun.star.beans.PropertyValue
Dim filterProps(1) As New com.sun.star.beans.PropertyValue

monDocument = thisComponent
filterProps(0).Name = "PageRange"
filterProps(0).Value = "6-7;10"
filterProps(1).Name = "CompressMode"
filterProps(1).Value = 0
propFich(0).Name = "FilterName"
propFich(0).Value = "writer_pdf_Export"
propFich(1).Name = "FilterData"
propFich(1).Value = filterProps()
adresseDoc = convertToURL("C:\Docs OpenOffice\docpartx.pdf")
monDocument.storeToURL(adresseDoc, propfich())
End Sub
```

Importer, exporter au format CSV

Les formats CSV (de l'anglais *Comma Separated Values*) sont des formats en texte pur dont chaque ligne comporte plusieurs champs, séparés les uns des autres avec une convention particulière. Par exemple, le séparateur de champ sera une virgule ou un caractère de tabulation. Chaque ligne est l'équivalent d'un enregistrement au sens des bases de données. Les fichiers au format CSV servent souvent de format intermédiaire entre deux applications ayant des formats de fichier incompatibles.

Le filtre CSV nécessite 5 paramètres qu'il faut compacter dans une simple chaîne de caractères afin de pouvoir l'affecter à `FilterOptions`.

Les paramètres du filtre CSV

Paramètre 1 : le séparateur de champ

Dans le cas général, le séparateur de champ est un caractère. Néanmoins, seule sa valeur numérique ASCII sera utilisée, et ceci sous forme de chaîne de caractères.

Le caractère Tabulation a pour valeur 9, le caractère Virgule a pour valeur 44. Pour tout caractère imprimable, une ligne Basic peut nous donner sa valeur ASCII. Par exemple, pour une virgule :

```
print ASC(",")
```

Si plusieurs séparateurs sont acceptables, on les séparera par le caractère barre de fraction. Par exemple, tabulation et virgule donneront un paramètre :

```
P1 = "9/44"
```

Si le séparateur est constitué par plusieurs caractères consécutifs, on ajoute à droite la séquence `MRG` ainsi :

```
' séparateur $$
P1 = "36/36/MRG"
```

Enfin, au lieu d'utiliser un caractère séparateur, on peut fixer le nombre de caractères de chaque champ : on parle de champs fixes. Le paramètre s'écrit :

```
P1 = "FIX"
```

Paramètre 2 : le délimiteur de champ texte

Un champ texte pouvant comporter divers caractères, il est encadré à gauche et à droite par un caractère ne se trouvant pas dans le texte. Habituellement, on emploie le caractère guillemets `"` (valeur 34) ou le caractère apostrophe `'` (valeur 39).

Le paramètre 2 contient la valeur décimale du caractère délimiteur de texte.

```
P2 = "39"
```

Paramètre 3 : le jeu de caractères utilisé

Comme le fichier CSV ne contient que des caractères ASCII, la représentation des caractères nationaux dépend du jeu de caractères utilisé. Le paramètre 3 contient une chaîne de caractères désignant le jeu employé. Malheureusement, la dénomination des jeux accepta-

bles n'est pas documentée ; quelques essais seront nécessaires. D'après des échanges dans des forums, ces valeurs existent pour OOo version 1.1 :

```
P3 = "STANDARD"
P3 = "IBMPC_850" ou "IBM_850" pour le jeu Dos occidental
```

Paramètre 4 : première ligne à traiter

Souvent, un fichier CSV comporte une ou plusieurs ligne(s) d'en-tête avant les lignes de données. Ce paramètre précise le numéro de la première ligne à traiter, la première ligne du fichier ayant le numéro 1. Exemple :

```
P4 = "2"
```

Paramètre 5 : format de chaque colonne

Il est nécessaire de préciser le format des données pour chaque champ. Le paramètre est alors constitué d'une séquence de caractères comportant :

1. le rang du champ (le premier champ a pour rang 1) ;
2. un caractère `/` ;
3. un nombre indiquant le format du champ ;
4. un caractère `/` avant la description de format du champ suivant.

Les valeurs de format possibles sont les suivantes :

- 1 standard (en importation, Calc décide du format) ;
- 2 texte ;
- 3 MM/DD/YY (mois, jour, an, deux chiffres chacun) ;
- 4 DD/MM/YY (jour, mois, an, deux chiffres chacun) ;
- 5 YY/MM/DD (an, mois, jour, deux chiffres chacun) ;
- 9 ignorer ce champ ;
- 10 nombre à l'américaine : point décimal et virgule comme séparateur de milliers.

Le format 10 a priorité sur un format de nombre national.

Exemple pour quatre champs :

```
P5 = "1/1/2/5/3/2/4/2"
```

Dans le cas particulier d'un format `FIX` (voir paramètre 1), le paramètre 5 contient, pour chaque champ successivement, la position du premier caractère de ce champ, à partir de la position zéro, un caractère `/` et la valeur de format. Exemple :

```
P5 = "0/1/8/5/18/2/45/2"
```

Constitution de la valeur de FilterOptions

La valeur de `FilterOptions` est donc une chaîne de caractères constituée par la concaténation des cinq paramètres, séparés par une virgule, soit :

```
propFich(1).Name = "FilterOptions"
propFich(1).Value = _
    P1 & "," & P2 & "," & P3 & "," & P4 & "," & P5
```

Bien entendu, nous avons employé des variables intermédiaires pour clarifier un peu. On aurait pu écrire par exemple :

```
propFich(1).Value = "9,39,STANDARD,2,1/1/2/5/3/2/4/2"
```

Importer, exporter du texte pur

Par texte pur, nous entendons un texte ordinaire, sans aucune marque de formatage.

Nous donnerons quelques exemples de filtres. Ils sont apparus dans des messages sur le forum de langue anglaise OOoForum (http://www.oooforum.org/) et n'ont pas été vérifiés.

Charger un fichier MS-DOS :

```
propFich(0).Name = "FilterName"
propFich(0).Value = "Text (encoded)"
propFich(1).Name = "FilterOptions"
propFich(1).Value = "IBM_850,CRLF,,"
```

Sur un système MS-Windows, ce filtre permet d'écrire un fichier texte encodé ANSI :

```
propFich(0).Name = "FilterName"
propFich(0).Value = "Text"
```

Ce filtre permet d'écrire un fichier texte encodé en UTF8 :

```
propFich(0).Name = "FilterName"
propFich(0).Value = "Text (encoded)"
propFich(1).Name = "FilterOptions"
propFich(1).Value = "UTF8, LF"
```

Imprimer un document

Nous décrirons ici les mécanismes d'impression communs à tous les documents. Les particularités propres à certains documents sont traitées dans les chapitres qui leur sont con-

sacrés. Les exemples utilisés ici sont disponibles à l'identique dans les fichiers du zip téléchargeable :

- `Code10-01.sxw` pour Writer
- `Code10-02.sxc` pour Calc
- `Code10-03.sxd` pour Draw

L'objet Printer

Tout document dispose normalement d'une imprimante pour son impression. Le document expose les caractéristiques de l'imprimante dans l'objet `Printer`. Ce dernier est un tableau (`Array`) contenant diverses propriétés listées dans le tableau 10-7, qui est suivi de quelques explications complémentaires.

Tableau 10–7 Propriétés du descripteur d'imprimante

Propriété	Type	Signification
`Name`	`String`	Le nom de la file d'attente de l'imprimante
`PaperOrientation`	`Integer`	Orientation du papier ; constante nommée, voir explications.
`PaperFormat`	`Integer`	Format du papier ; constante nommée, voir explications.
`PaperSize`	`Object`	Taille du papier, largeur et hauteur en 1/100 de mm.
`IsBusy`	`Boolean`	`True` si l'imprimante est occupée (au niveau de la mise en file d'attente du travail).
`CanSetPaperOrientation`	`Boolean`	`True` si on peut changer l'orientation du papier.
`CanSetPaperFormat`	`Boolean`	`True` si on peut changer le format de papier.
`CanSetPaperSize`	`Boolean`	`True` si on peut imposer une taille quelconque du papier.

L'orientation du papier ne peut prendre que deux valeurs, sous la forme de constantes nommées :

```
com.sun.star.view.PaperOrientation.PORTRAIT
com.sun.star.view.PaperOrientation.LANDSCAPE
```

Attention à la casse ! Les constantes nommées doivent être écrites en respectant les majuscules et minuscules.

Plusieurs formats standardisés de papier sont reconnus. Les constantes nommées sont de la forme :

```
com.sun.star.view.PaperFormat.A4
```

Les différents formats de papier possibles sont :

```
A3 A4 A5 B4 B5 LETTER LEGAL TABLOID USER
```

En Europe, nous utilisons essentiellement les formats A3, A4 et A5. Le format `USER` est indiqué lorsque le format ne correspond à aucun de ceux cités. Dans ce dernier cas seulement, la propriété `PaperSize` est significative ; elle se compose de deux éléments `Width` et `Height` indiquant la largeur et la hauteur en twips. Un twip représente 1/20 de point, qui représente lui-même 1/72 de pouce. Un pouce représentant environ 25,4 mm, cela donne 1 twip = 1,76 centième de millimètre.

Cet exemple va lister les propriétés d'imprimante. Il est expliqué juste après.

```
rem Code10-01.sxw   bibli : Imprimer Module1
Option Explicit

Sub LirePropImprimante()
Dim monDocument As Object, imprimante As Variant
Dim papVal As Integer, papTaille As Object
Dim liste As String, cr As String

Const twipsParMm = 56.7
cr = chr(13) ' caractère de fin de ligne
monDocument = ThisComponent
imprimante = monDocument.Printer
printProps(imprimante) ' lister les noms de propriétés

liste = "Nom : " & getPropVal(imprimante, "Name") & cr & "Occupée : " & _
   getPropVal(imprimante, "IsBusy") & cr & _
  "Orientation de papier modifiable : " & _
   getPropVal(imprimante, "CanSetPaperOrientation") & cr & _
  "Format de papier modifiable : " & _
   getPropVal(imprimante, "CanSetPaperFormat") & cr & _
  "Taille de papier modifiable : " & getPropVal(imprimante, "CanSetPaperSize") & cr

papTaille = getPropVal(imprimante, "PaperSize")
papVal = getPropVal(imprimante, "PaperOrientation")
if papVal = com.sun.star.view.PaperOrientation.PORTRAIT then
  liste = liste & "Orientation Portrait" & cr
else
  liste = liste & "Orientation Paysage" & cr
end if

papVal = getPropVal(imprimante, "PaperFormat")
Select Case papVal
Case com.sun.star.view.PaperFormat.A4
  liste = liste & "Format A4"
Case com.sun.star.view.PaperFormat.LETTER
  liste = liste & "Format Letter"
Case com.sun.star.view.PaperFormat.USER
  liste = liste & "Format inconnu : "

Case Else
  liste = liste & "Autre format connu"
```

```
end Select
  liste = liste & cr & _
  "Hauteur : " & Format(papTaille.Height/twipsParMm, "0.#") & " mm" & cr & _
  "Largeur : " & Format(papTaille.Width/twipsParMm,  "##0.#") & " mm"
MsgBox(liste, 0, "Capacités de l'imprimante")
End Sub
```

La propriété `Printer` de l'objet document est un tableau de propriétés. La difficulté est que l'ordre des propriétés dépend de l'implémentation. Pour la contourner, nous utiliserons deux routines utilitaires qui sont détaillées à l'annexe B.

La première, `printProps`, balaie le tableau de propriétés et pour chacune récupère son nom pour l'afficher, avec l'index dans le tableau.

La macro principale va ensuite constituer un texte dans la variable `liste`, avec une ligne pour chaque propriété. Pour récupérer la valeur d'une propriété particulière, il faut la rechercher dans le tableau de propriétés. C'est ce que fait la deuxième routine utilitaire `getPropVal`, qui récupère la valeur de la propriété si elle la trouve ; sinon, elle affiche un message d'erreur.

Dans le document où se trouve la macro principale, ouvrez le menu Fichier > Paramétrage de l'imprimante... pour changer les caractéristiques d'impression. Relancez la macro, vous devriez pouvoir ainsi passer par les différents cas prévus.

Changer la configuration d'impression

Il est possible de changer d'imprimante, et dans la mesure où elle le permet, vous pouvez imposer les valeurs de ses propriétés. Dans cet exemple, nous changeons d'imprimante et imposons le format de papier *Letter* en orientation *Paysage*.

```
rem Code10-01.sxw   bibli : Imprimer Module2
Option Explicit

Sub ChangerImprimante()
Dim monDocument As Object, imprimante As Variant
Dim nomImpr As String
monDocument = ThisComponent
imprimante = monDocument.Printer

nomImpr = InputBox("Nom de l'imprimante ?")
setPropVal(imprimante, "Name", nomImpr)
setPropVal(imprimante, "PaperFormat", com.sun.star.view.PaperFormat.LETTER)
setPropVal(imprimante, "PaperOrientation", _
                        com.sun.star.view.PaperOrientation.LANDSCAPE)

monDocument.Printer = imprimante
LirePropImprimante() ' lister les noms de propriétés
End Sub
```

Nous modifions le tableau de propriétés avec la routine utilitaire `setPropVal` décrite à l'annexe B. Elle recherche la propriété du nom donné en argument, et la modifie. Une fois le tableau modifié, il est copié dans la propriété `Printer` de l'objet document. Si vous répondez à la question avec un nom d'imprimante inexistante, le document reviendra sur l'imprimante par défaut.

Si votre imprimante accepte les changements de format et d'orientation de page, le format de page en cours sera modifié. Vous pouvez le vérifier en ouvrant le menu Format > Page... onglet Page.

Pour une imprimante réseau, suivant la configuration en place, il peut être nécessaire d'entourer le nom de l'imprimante avec les symboles < et > comme ceci :

```
nomImpr = "<ImprFinancier>"
```

Lancer l'impression

L'impression d'un document se fait au moyen de sa méthode `print`, qui a pour argument un tableau contenant les options d'impression désirées (voir tableau 10-8).

Tableau 10–8 Options d'impression

Propriété	Type	Signification
`FileName`	`String`	Le nom du fichier d'impression, si on utilise cette possibilité.
`CopyCount`	`Integer`	Nombre d'exemplaires à imprimer. Par défaut : 1
`Collate`	`Boolean`	`True` si on imprime les copies multiples document par document (valeur par défaut). `False` si on imprime toutes les copies d'une page avant de passer à la page suivante.
`Pages`	`String`	Indique les pages à copier, comme dans l'interface utilisateur. Exemple : 1-4;10;15-18 Par défaut : toutes les pages sont imprimées.
`Wait`	`Boolean`	`True` pour attendre la fin de la mise en file d'attente d'impression

Un petit exemple :

```
rem Code10-01.sxw   bibli : Imprimer Module3
Option Explicit

Sub ImprimerPartie()
Dim monDocument As Object
Dim Props(0) As New com.sun.star.beans.PropertyValue
monDocument = ThisComponent
Props(0).Name = "Pages"
```

```
Props(0).Value = "3;15"
monDocument.Print(Props())
End Sub
```

Si vous exécutez cette macro sur un document d'une seule page, il ne se passera rien puisqu'aucune des pages demandées n'existe.

Si on se contente de toutes les valeurs par défaut, il faut néanmoins fournir un tableau sans dimension, déclaré ainsi :

```
Dim Props()
```

Note : si le document doit être fermé immédiatement après le lancement de l'impression, il est nécessaire de mettre à `True` la propriété `Wait`.

Les informations du document

Rappelons que l'adresse du fichier d'un document ouvert est disponible dans sa propriété `Location` :

```
print monDocument.Location
```

Tout document bureautique OpenOffice.org contient des informations concernant le contexte dans lequel il a été rédigé, comme son auteur ou sa date de création. Il s'agit des informations accessibles dans l'interface utilisateur par le menu Fichier > Propriétés. La structure de la propriété `documentInfo` contient ces informations, accessibles en ajoutant un point suivi du nom de l'élément de structure. Voici comment obtenir l'auteur du document :

```
mesInfos = monDocument.documentInfo
print mesInfos.author
```

Le tableau 10-9 donne la liste des principales propriétés accessibles par l'intermédiaire de cet objet. Le SDK (`com.sun.star.document`, service `DocumentInfo`) est assez confus sur cette page, avec des incohérences entre les versions 1.1, la version 2.0.0, et la réalité de l'API de chaque version. Nous décrivons ce qui est effectivement disponible.

Tableau 10–9 Informations sur le document

Élément	Type	Signification
`Author`	`String`	Auteur initial du document.
`CreationDate`	`Object`	Date de création du document, voir exemple.
`Description`	`String`	Champ `Commentaire` des propriétés du document ; peut comporter plusieurs lignes.

Tableau 10–9 Informations sur le document

Élément	Type	Signification
Keywords	String	Mots-clés.
MIMEType	String	Type MIME du document (en lecture seule). Il permet de distinguer un document OpenOffice.org version 1.1 d'un document au format Oasis OpenDocument. Pour un document Writer, chargé avec la version 2.0 d'OpenOffice.org, on obtient selon le cas : application/vnd.sun.xml.writer application/vnd.oasis.opendocument.text
ModifiedBy	String	Nom du dernier utilisateur ayant enregistré le document.
ModifyDate	Object	Date de dernière modification.
PrintedBy	String	Nom du dernier utilisateur ayant imprimé le document.
PrintDate	Object	Date de dernière impression. N'existe que si le document a été imprimé.
Subject	String	OOo 1.1 : propriété inexistante. OOo 2.0.0 : champ Sujet des propriétés du document.
Template	String	Nom du modèle utilisé.
TemplateFileName	String	Chemin et nom du fichier modèle utilisé.
TemplateDate	Object	Date du modèle utilisé.
Theme	String	OOo 1.1 : champ Sujet des propriétés du document. OOo 2.0.0 : propriété inexistante.
Title	String	Titre du document.
EditingCycles	Integer	Numéro de version
EditingDuration	Long	Durée d'édition, exemple: 2562600 pour 2H56M26S

Les champs de date sont des structures `com.sun.star.util.DateTime` contenant les valeurs numériques de jour, mois, année, heure, minute, seconde et centième de seconde. Ainsi, pour afficher la date de création du document, nous utiliserons :

```
rem Code10-03.sxd   bibli : DocInfos Module1
Option Explicit

Sub DateDuDocument()
Dim uneDate As Variant
Dim Chaine As String

uneDate = ThisComponent.DocumentInfo.CreationDate
With uneDate ' pour éviter de répéter uneDate
```

```
  chaine = .Day & "/" & .Month & "/" & .Year & " " & .Hours & ":" & _
           .Minutes & ":" & .Seconds & "." & .HundredthSeconds
End With
MsgBox(chaine)
End Sub
```

Les instructions `With... End With` nous évitent de répéter la variable `uneDate` devant chaque élément de la structure, mais le point doit cependant être écrit.

Vous pouvez modifier par macro chacune des informations, sauf `MIMEType`, qui est en lecture seule. Voici comment en modifier plusieurs à la fois :

```
rem Code10-01.sxw   bibli : DocInfos Module3
Option Explicit

Sub ChangerPropsDoc()
Dim monDocument As Object, lesInfos As Object


monDocument = ThisComponent
lesInfos = monDocument.DocumentInfo
With lesInfos
  .Author = "Zoé"
  .Description = "comment changer les informations de document"
  .ModifiedBy = "Laura"
  .Title = "Essai de macro"
  .EditingCycles = 456
End With
On Error Resume Next
monDocument.DocumentInfo = lesInfos
On Error Goto 0
End Sub
```

Pour recopier la structure `lesInfos`, nous avons désactivé le traitement d'erreurs Basic, sinon il rouspéterait que « la propriété est en lecture seule », à cause de `MIMEType`.

L'objet `DocumentInfo` possède également des méthodes pour manipuler les champs d'information `info1`, `info2`, `info3` et `info4` accessibles par l'utilisateur dans les propriétés du document. Ces champs peuvent être un moyen de stocker des informations entre deux sessions du document.

```
rem Code10-03.sxd   bibli : DocInfos Module2
Option Explicit

Sub ChampsUtilisateur()
dim mesInfos as object
dim NbInfos as long, i as long
mesInfos=thisComponent.DocumentInfo
NbInfos=mesInfos.getUserFieldCount
```

```
for i=0 to nbInfos-1
  print mesInfos.getUserFieldName(i) & " = " & mesInfos.getUserFieldValue(i)
next i

for i=0 to nbInfos-1
  mesInfos.setUserFieldName(i,"Titre du champ "+cstr(i+1))
  mesInfos.setUserFieldValue(i, "Contenu du champ utilisateur "+cstr(i+1))
next i

for i=0 to nbInfos-1
  print mesInfos.getUserFieldName(i) & " = " & _
        mesInfos.getUserFieldValue(i)
next i
End Sub
```

La macro commence par récupérer le nombre de champs par la méthode `getUserFieldCount`. Une première boucle en affiche les noms successifs par `getUserFieldName`, ainsi que les valeurs correspondantes par `getUserFieldValue`.

La boucle suivante change la dénomination de ces champs par `setUserFieldName`, ainsi que leur contenu par `setUserFieldValue`.

La dernière boucle, identique à la première, permet de vérifier la prise en compte des changements. Cette vérification peut également être faite en allant dans le menu Fichier > Propriétés > Utilisateur.

La langue utilisée par défaut dans le document est obtenue par la propriété `Locale` du document. C'est une structure contenant notamment deux chaînes de caractères : la langue et la variante de pays :

```
print thisComponent.CharLocale.Language
print thisComponent.CharLocale.Country
```

Informations spécifiques à un document Writer ou HTML

Un document texte expose quelques propriétés intéressantes le concernant :

- `CharacterCount`, de type `Long` : le nombre total de caractères ;
- `WordCount`, de type `Long` : le nombre total de mots ;
- `ParagraphCount`, de type `Long` : le nombre total de paragraphes ;
- `WordSeparator`, de type `String` : les caractères séparateurs de mots (en plus de tabulation, espace, fin de paragraphe, fin de ligne).

Exemple :

```
print thisComponent.WordCount
```

Configuration d'affichage d'un document

Chaque type de document possède des propriétés d'affichage accessibles à partir de l'objet contrôleur du document. La manière d'y accéder peut varier, et les propriétés sont pour la plupart spécifiques.

Les propriétés du zoom d'affichage sont, elles, identiques. Elles sont au nombre de deux :

- `ZoomType`, de type `Integer`, qui reçoit une constante nommée de la forme `com.sun.star.view.DocumentZoomType.OPTIMAL` et dont les valeurs possibles sont listées dans le tableau 10-10 ;
- `ZoomValue`, de type `Integer`, qui reçoit le facteur de zoom en pourcentage. Cette propriété est utilisée quand `ZoomType` vaut `BY_VALUE`.

Tableau 10–10 Constantes de facteur de zoom

Constante	Signification
`OPTIMAL`	Optimal.
`PAGE_WIDTH`	Largeur de page.
`ENTIRE_PAGE`	Page entière.
`PAGE_WIDTH_EXACT`	Page entière.
`BY_VALUE`	Facteur de zoom selon `ZoomValue`.

Exemple, sur un document Writer :

```
rem Code10-01.sxw   bibli : Config Module1
Option Explicit

Sub Zoomer()
Dim monDocument As Object, conf As Object
Dim z As Integer
monDocument = thisComponent
conf = monDocument.CurrentController.ViewSettings
z = InputBox("Facteur de zoom, en %, ou zéro")
if z > 0 then
  conf.ZoomValue = z
  conf.ZoomType = com.sun.star.view.DocumentZoomType.BY_VALUE
else
  conf.ZoomType = com.sun.star.view.DocumentZoomType.PAGE_WIDTH
end if
End Sub
```

Configuration d'un document

Le service `DocumentSettings` de chaque type de document fournit quelques informations intéressantes. Pour les obtenir, nous devons invoquer le service à partir de l'objet document, par exemple pour un document Writer :

```
rem Code10-01.sxw   bibli : Imprimer Module3
Option Explicit

Sub ConfigImpression()
Dim monDocument As Object, conf As Object, servConfig As String
monDocument = ThisComponent
servConfig = "com.sun.star.text.DocumentSettings"
conf = monDocument.createInstance(servConfig)
print conf.PrinterName
End Sub
```

Le nom du service dépend du document, voir le tableau 10-11.

Tableau 10–11 Service de configuration de document

Document	Nom du service
Writer	`com.sun.star.text.DocumentSettings`
Calc	`com.sun.star.comp.SpreadsheetSettings`
Draw	`com.sun.star.drawing.DocumentSettings`
Impress	`com.sun.star.presentation.DocumentSettings`

Quelques propriétés communes sont listées au tableau 10-12. Nous signalons dans les chapitres consacrés à chaque type de document les propriétés de configurations qui leur sont spécifiques et particulièrement notables.

Tableau 10–12 Propriétés communes de configuration

Propriété	Type	Signification
`PrinterName`	`String`	Nom de l'imprimante utilisée par le document.
`SaveVersionOnClose`	`Boolean`	`True` pour créer une nouvelle version à la fermeture du document modifié.
`UpdateFromTemplate`	`Boolean`	`True` pour mettre à jour le document si son modèle a évolué.
`FieldAutoUpdate`	`Boolean`	`True` pour que les champs soient mis à jour automatiquement.

Conclusion

Le premier chapitre de cette partie a été l'occasion d'utiliser l'API d'OpenOffice.org pour manipuler les documents. Ouverture, fermeture, import/export... nous avons agi indépendamment de leur nature.

Le chapitre suivant expose et illustre l'API propre aux documents issus du traitement de texte Writer.

11

Les documents Writer

Dans ce chapitre, nous allons commencer par exposer les activités les plus courantes dans la manipulation par macro des documents Writer : lire et écrire du texte dans le document, formater du texte, rechercher et remplacer du texte. Nous aborderons ensuite des domaines plus spécialisés comme les tableaux et les cadres, les styles, les signets. Vous retrouverez parfois exactement les mêmes concepts dans un contexte différent. Ceci provient de la conception globale de l'interface de programmation (l'API) d'OpenOffice.org et vous facilitera l'apprentissage.

> API **Référence sur Writer (en anglais)**
>
> La documentation de l'API est décrite dans le chapitre 7 du « Developer's Guide ». Il comporte de nombreux liens hypertextes vers les descriptions des objets. Voir l'annexe A pour une brève présentation.

Imprimer

L'impression d'un document Writer n'a aucune particularité spéciale. Reportez-vous au chapitre 10 où sont décrits les mécanismes d'impression d'un document.

L'objet Text

Pour travailler sur le texte d'un document Writer, nous avons besoin de préciser de quel texte il s'agit. En effet, un document Writer contient différents objets : le texte ordinaire, les en-têtes et pieds de page, ou des cadres contenant du texte. Le texte principal est l'objet `Text`, obtenu à partir du document :

```
Dim monDocument As Object, monTexte As Object
monDocument = ThisComponent
monTexte = monDocument.Text
```

La variable `monTexte` nous permet de manipuler l'objet `Text` du document. Elle possède une propriété ayant pour nom `String`, qui représente l'ensemble des caractères du texte. Cette propriété fournit une chaîne de caractères. Le codage suivant affiche les 1 000 premiers caractères du texte du document Writer.

```
rem Code11-01.sxw   bibli : EcritureTexte Module1
Option Explicit

Sub AfficherTexte()
Dim monDocument As Object, monTexte As Object
monDocument = ThisComponent
monTexte = monDocument.Text
' afficher les premiers caractères du texte
MsgBox Left(monTexte.String, 1000)
End Sub
```

Inversement, en affectant une chaîne de caractères à la propriété `String`, nous changeons le texte entier du document :

```
rem Code11-01.sxw   bibli : EcritureTexte Module2
Option Explicit

Sub EcraserTexte()
Dim monDocument As Object, monTexte As Object
monDocument = ThisComponent
monTexte = monDocument.Text

monTexte.String = "J'ai écrasé tout le texte !"
End Sub
```

Si vous avez exécuté cette macro, il ne vous reste plus qu'à cliquer sur le bouton d'annulation dans la fenêtre du document, afin de récupérer l'ancien texte.

Cette méthode de lecture et d'écriture est simple, mais très limitative. D'une part, le texte manipulé doit avoir moins de 65 536 caractères pour qu'il soit contenu dans une variable de type `String`. D'autre part, nous ne pouvons pas imposer un formatage particulier de certaines zones du texte. Pour avoir plus de possibilités, nous allons utiliser un objet curseur, qui permet de préciser la zone de texte que nous voulons lire ou modifier.

Le curseur d'écriture

Il faut bien comprendre qu'il existe deux types de curseurs :

- le curseur visible, celui que vous voyez avancer quand vous tapez du texte dans Writer (il sera décrit plus loin) ;
- le curseur d'écriture, invisible, qui est utilisé par la macro pour désigner le texte qu'elle manipule.

La position courante d'un curseur d'écriture n'est absolument pas liée à la position du curseur visible.

On peut créer un ou plusieurs curseur(s) d'écriture grâce à la méthode `createTextCursor` de l'objet `Text` du document.

```
Dim monCurseur As Object
monCurseur = monTexte.createTextCursor
```

Le curseur peut, soit indiquer un point d'insertion dans le texte, soit délimiter une zone de sélection dans le texte (comme un curseur visible). À la création, le curseur est un point d'insertion placé au début du texte. Il avancera en fonction des caractères écrits par la macro. Pour insérer du texte à un endroit quelconque du texte existant, il nous faut apprendre à déplacer ce curseur.

L'objet curseur comporte un très grand nombre de propriétés et de méthodes. Grâce à lui, il est possible d'obtenir des informations ou de modifier le caractère, le paragraphe et bien d'autres éléments auxquels il est lié.

Déplacer le curseur d'écriture

L'objet curseur dispose de plusieurs fonctions permettant de le déplacer. Elles renvoient presque toutes le résultat :

- `True` si l'action a pu être réalisée,
- `False` dans le cas contraire.

En pratique, on utilise rarement le résultat de ces fonctions, et on les utilise comme des méthodes de type `Sub`.

Les fonctions de déplacement ont toutes un argument booléen, que nous désignerons par `SEL` et qui a l'effet suivant :

- `SEL = False` : le curseur se déplace (comme la barre verticale du curseur visible),
- `SEL = True` : le curseur se déplace en étendant la sélection (c'est le même effet qu'une sélection progressive du curseur visible en faisant glisser la souris).

Et voici un petit exemple :

```
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
' ici le curseur est au début du texte
' aller au paragraphe suivant
monCurseur.gotoNextParagraph(False)
' sélectionner les 3 caractères suivants
monCurseur.goRight(3,True)
```

Le tableau 11-1 liste les fonctions de déplacement de curseur. La méthode `gotoRange` est un peu particulière : elle utilise en premier argument une zone de texte (anglais : `TextRange`), qui est un repérage dans le texte. Cette zone peut provenir de différents objets, en général un autre curseur ou une ancre (anglais : `Anchor`). Nous donnerons un exemple un peu plus loin.

Tableau 11–1 Déplacement du curseur d'écriture

Méthode	Effet sur le curseur
`goRight(n,SEL)`	Déplacer de n caractères à droite.
`goLeft(n,SEL)`	Déplacer de n caractères à gauche.
`gotoStart(SEL)`	Déplacer au début du texte entier. Cette méthode ne renvoie pas de résultat.
`gotoEnd(SEL)`	Déplacer à la fin du texte entier. Cette méthode ne renvoie pas de résultat.
`gotoRange(zone, SEL)`	Positionner le curseur sur une zone donnée.
`gotoStartOfParagraph(SEL)`	Déplacer au début du paragraphe en cours.
`gotoEndOfParagraph(SEL)`	Déplacer à la fin du paragraphe en cours.
`gotoNextParagraph(SEL)`	Déplacer au début du paragraphe suivant.
`gotoPreviousParagraph(SEL)`	Déplacer au début du paragraphe précédent.
`gotoNextWord(SEL)`	Déplacer au début du mot suivant.
`gotoPreviousWord(SEL)`	Déplacer au début du mot précédent.
`gotoStartOfWord(SEL)`	Déplacer au début du mot courant.
`gotoEndOfWord(SEL)`	Déplacer à la fin du mot courant.
`gotoNextSentence(SEL)`	Déplacer au début de la phrase suivante.
`gotoPreviousSentence(SEL)`	Déplacer au début de la phrase précédente.
`gotoStartOfSentence(SEL)`	Déplacer au début de la phrase courante.
`gotoEndOfSentence(SEL)`	Déplacer à la fin de la phrase courante.

Le tableau 11-2 liste les fonctions de l'objet curseur qui indiquent si le curseur est à un endroit remarquable. Elles renvoient toutes un résultat booléen : `True` si la réponse est affirmative, `False` sinon. Ces fonctions n'utilisent pas d'argument, donc les parenthèses peuvent être omises en Basic.

```
if monCurseur.isEndOfWord then print "Fin de mot"
```

Tableau 11–2 Fonctions testant si le curseur est à une position remarquable

Fonction	Test effectué
`isStartOfParagraph()`	Le curseur est-il au début d'un paragraphe ?
`isEndOfParagraph()`	Le curseur est-il à la fin d'un paragraphe ?
`isStartOfWord()`	Le curseur est-il au début du mot en cours ?
`isEndOfWord()`	Le curseur est-il à la fin du mot en cours ?
`isStartOfSentence()`	Le curseur est-il en début de phrase ?
`isEndOfSentence()`	Le curseur est-il en fin de phrase ?
`isCollapsed()`	Le curseur est-il un simple point d'insertion ?

ALLER PLUS LOIN La notion de mot

Les fonctions de déplacement et de test concernant les mots et les phrases peuvent donner des résultats surprenants. Writer considère qu'un mot se termine avec un espace, une fin de paragraphe, un retour forcé à la ligne, ou un des caractères de la chaîne `WordSeparator`, qui est une propriété du document. De plus, le déplacement du curseur de mot s'arrête sur des caractères comme « + / , ».

La fonction `isCollapsed` mérite une explication : elle renvoie `False` si le curseur est étendu pour une sélection. Dans ce dernier cas, le curseur s'étend d'une position à une autre dans le texte. On peut revenir à un curseur ponctuel en utilisant une des deux méthodes de l'objet curseur qui sont listées ci-dessous. Ces méthodes n'ont pas d'argument.

```
' réduire le curseur sur sa position de début
monCurseur.collapseToStart

' réduire le curseur sur sa position de fin
monCurseur.collapseToEnd
```

Autres initialisations d'un curseur

Dans certains cas, il est intéressant d'utiliser un autre curseur d'écriture qui soit initialisé aux valeurs du curseur actuel. On utilise pour cela la méthode `createTextCursorByRange` de l'objet `Text`. Dans cet exemple, `curseur2` reprend la position et la sélection dans le texte mémorisées par `monCurseur` :

```
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, curseur2 As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.goRight(3,True)
curseur2 = monTexte.createTextCursorByRange(monCurseur)
```

Les objets `curseur2` et `monCurseur` peuvent être déplacés ou modifiés indépendamment l'un de l'autre.

On peut aussi créer un autre curseur à partir du premier en précisant qu'il doit se positionner à la fin de la zone de texte désignée par `monCurseur`. Il suffit de modifier la dernière ligne ainsi :

```
curseur2 = monTexte.createTextCursorByRange(monCurseur.End)
```

Pour créer `curseur2` au début de la zone de `monCurseur` on écrirait :

```
curseur2 = monTexte.createTextCursorByRange(monCurseur.Start)
```

Nous pouvons aussi positionner un curseur existant sur une zone de texte, en employant la méthode `gotoRange` du curseur, par exemple :

```
curseur2.gotoRange(monCurseur.End)
```

Notez que les deux méthodes `createTextCursorByRange` et `gotoRange` utilisent comme argument un objet zone de texte (anglais *TextRange*) ; dans les exemples, le curseur ne sert qu'à fournir un tel objet.

Lire une zone de texte

Un moyen pour lire du texte est de sélectionner une zone grâce aux déplacements de curseur, puis récupérer le texte ainsi sélectionné avec la propriété `String` de l'objet curseur. Le document reste inchangé. Dans cet exemple, nous allons récupérer puis afficher le troisième paragraphe du document.

```
rem Code11-01.sxw   bibli : EcritureTexte Module3
Option Explicit

Sub AfficherTexteSelection()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
```

```
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(True)
' afficher le texte du troisième paragraphe
MsgBox monCurseur.String
End Sub
```

La propriété `String` est du type `String`. La sélection ne doit pas englober plus de 65 535 caractères.

ATTENTION Il y a String et String

Ici encore, le terme `String` est employé. Il s'agit d'une propriété qui appartient à un objet curseur, elle est différente de la propriété `String` de l'objet `Text`. Chaque type d'objet peut utiliser n'importe quel nom pour ses propriétés et méthodes. Pourquoi reprendre le même nom ? parce que, au fond, ces propriétés désignent des concepts similaires : une chaîne de caractères représentant un texte. Et une chaîne de caractères, en anglais, cela s'appelle `String`. Ce nom est donc aussi utilisé en Basic et dans d'autres langages de programmation pour désigner le type de variable « chaîne de caractères ».

Insérer du texte

Avec la propriété String

En affectant une chaîne de caractères à la propriété `String` du curseur, nous allons insérer cette chaîne à l'emplacement du curseur :

```
rem Code11-01.sxw   bibli : EcritureTexte Module4
Option Explicit

Sub AjouterTexte()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextWord(False)
' ajouter un texte après le 1er mot du 3ème paragraphe
monCurseur.String = "*****"
End Sub
```

Si nous avions utilisé l'argument `True` dans la méthode `gotoNextWord`, le premier mot du paragraphe aurait été remplacé par la nouvelle chaîne.

Cette méthode a des limitations : la taille de la chaîne de caractères insérée est bornée, on ne peut pas modifier le formatage, il faut déplacer le curseur par programmation après chaque écriture de chaîne sinon la suivante écrasera la précédente.

> **À RETENIR Caractères spéciaux**
>
> Dans la chaîne de caractères insérée, un caractère `chr(10)` insère un saut de ligne et un caractère `chr(9)` insère une tabulation.

Avec la méthode insertString

Avec cette méthode, le curseur d'écriture se positionne à la fin de la chaîne de caractères écrite, ce qui permet d'en insérer une autre à la suite.

```
rem Code11-01.sxw   bibli : EcritureTexte Module5
Option Explicit

Sub InsererTexte()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextWord(False)
' ajouter un texte après le 1er mot du 3ème paragraphe
monTexte.insertString(monCurseur, "AAAAA", false)
monTexte.insertString(monCurseur, "bbbb", false)
End Sub
```

La méthode `insertString` de l'objet `Text` utilise trois arguments :

1. un objet curseur,
2. la chaîne de caractères à insérer,
3. une valeur booléenne.

Le troisième argument reçoit en général la valeur `False`. La valeur `True` est employée pour remplacer une zone préalablement sélectionnée avec le curseur. Après insertion, le curseur redevient ponctuel et positionné au début de l'ancienne sélection, c'est-à-dire juste *avant* le texte qui vient d'être inséré. Ce comportement n'est pas celui décrit dans la documentation API.

Insérer des caractères spéciaux

Dans la chaîne de caractères insérée avec la méthode `insertString`, un caractère `chr(10)` insère un saut de ligne et un caractère `chr(9)` insère une tabulation. La méthode `insertControlCharacter` de l'objet `Text` permet d'insérer d'autres caractères spéciaux.

```
rem Code11-01.sxw   bibli : EcritureTexte Module6
Option Explicit

Sub InsererCarControle()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
Dim special As Integer
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextWord(False)
monCurseur.gotoNextWord(False)
' insérer une fin de paragraphe après 2ème mot 3ème paragraphe
special = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
monTexte.insertControlCharacter(monCurseur, special, false)
monTexte.insertString(monCurseur, "AAAAA", false)
End Sub
```

Le premier argument de `insertControlCharacter` est un objet curseur. Le deuxième argument est une valeur numérique du type `Integer`, qui se définit par une constante nommée (tableau 11-3) de la forme :

```
com.sun.star.text.ControlCharacter.LINE_BREAK
```

Attention à la casse ! Les constantes nommées doivent être écrites en respectant les majuscules et minuscules.

Le troisième argument a la même signification que pour `insertString`.

Tableau 11–3 Constantes de caractères spéciaux

Constante	Signification
PARAGRAPH_BREAK	Insérer ici une marque de fin de paragraphe.
LINE_BREAK	Insérer ici un retour à la ligne (équivalent de la frappe Maj + Entrée).
HARD_HYPHEN	Insérer ici un tiret insécable (le mot ne doit jamais être coupé après ce tiret).
SOFT_HYPHEN	Insérer ici un tiret conditionnel (le mot peut être coupé à cet endroit).
HARD_SPACE	Insérer ici un espace insécable (les deux mots ne doivent pas être séparés).
APPEND_PARAGRAPH	Insérer un paragraphe à la fin de celui en cours et se positionner au début du nouveau paragraphe.

Insérer un saut de page ou de colonne

Le saut de page est une propriété de paragraphe. Vous pouvez définir un style de paragraphe comportant un saut de page. Sinon, vous pouvez insérer une marque de saut de page dans le paragraphe en cours grâce à la propriété `breakType` de l'objet curseur :

```
rem Code11-01.sxw   bibli : EcritureTexte Module7
Option Explicit

Sub InsererSautPage()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
Dim saut As Integer
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)

' insérer un saut de page AVANT ce 3ème paragraphe
saut = com.sun.star.style.BreakType.PAGE_BEFORE
monCurseur.breakType = saut
monTexte.insertString(monCurseur, "AAAAA", false)
End Sub
```

Dans le cas d'un document vierge ne contenant donc pas de paragraphe, il est nécessaire d'insérer une fin de paragraphe avant le saut de page. Le saut est une valeur numérique du type `Integer`, qui se définit par une constante nommée (tableau 11-4) de la forme :

```
com.sun.star.style.BreakType.NONE
```

Tableau 11–4 Constantes de saut de page

Constante	Signification
PAGE_BEFORE	Changer de page *avant* le paragraphe en cours.
PAGE_AFTER	Changer de page *après* le paragraphe en cours.
PAGE_BOTH	Changer de page *avant et après* le paragraphe en cours.
NONE	Supprimer le saut de page ou de colonne qui existe dans le paragraphe.
COLUMN_BEFORE	Changer de colonne *avant* le paragraphe en cours.
COLUMN_AFTER	Changer de colonne *après* le paragraphe en cours.
COLUMN_BOTH	Changer de colonne *avant et après* le paragraphe en cours.

Les sauts de type `COLUMN` sont utilisés dans un texte en colonnes. En effet, on peut avoir deux ou trois colonnes par page, et souhaiter changer de colonne sans obligatoirement changer de page.

Pour insérer des pages d'orientation différente, par exemple passer d'une orientation Portrait à Paysage puis revenir à Portrait, il faut utiliser des styles de pages différents. La méthode est développée plus loin dans la section traitant des styles.

Insérer le texte d'un autre document

Le curseur d'écriture expose la méthode `insertDocumentFromURL` qui permet d'insérer un texte provenant d'un autre document. Pour cet exemple, recopiez au bon endroit le fichier `Poeme.sxw` qui se trouve dans le zip téléchargeable, dans le même répertoire, et adaptez la valeur de la variable `adresseDoc`.

```
rem Code11-12.sxw   bibli : Standard Module1
Option Explicit

Sub InsererDocumentTexte()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, adresseDoc As String
Dim propFich() As New com.sun.star.beans.PropertyValue
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False) ' troisième paragraphe
monCurseur.gotoNextWord(True) ' sélectionner le premier mot

' insérer un document à la place du texte sélectionné
adresseDoc = convertToURL("C:\Docs OpenOffice\Poeme.sxw")
monCurseur.insertDocumentFromURL(adresseDoc, propFich())
End Sub
```

L'argument `propFich()` peut contenir les propriétés listées au chapitre 10 pour la méthode `LoadComponentFromURL`.

Vous remarquerez que les styles du document inséré sont importés dans le document hôte. En revanche, les en-têtes et bas de pages ne sont pas importés.

Supprimer des paragraphes

Il faut distinguer la suppression d'une marque de paragraphe et la suppression complète d'un paragraphe.

Supprimer une marque de paragraphe

Une marque de paragraphe est un caractère spécial dans le texte. On peut donc la supprimer comme on supprimerait un caractère du texte, ce qui aboutit à accoler le texte du paragraphe avec le paragraphe suivant. Notre exemple parcourt un texte complet et supprime les marques de paragraphe en remplaçant chacune par un espace.

```
rem Code11-11.sxw   bibli : Standard Module1
Option Explicit

Sub Supprimer_MarquesParagraphes()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur= monTexte.createTextCursor
monCurseur.gotoStart(false)
Do While monCurseur.gotoNextParagraph(false)
  monCurseur.goLeft(1, true)
  monCurseur.String = " "
Loop
End Sub
```

Le texte de l'exemple dans le zip téléchargeable est un poème dont chaque strophe est un paragraphe comportant un retour à la ligne pour chaque vers. Les paragraphes ont différents styles. Après exécution de la macro, les retours à la ligne sont conservés, mais à chaque suppression de fin de paragraphe, la strophe correspondante a pris le style de la strophe suivante.

Supprimer tout un paragraphe

L'objet texte d'un document Writer est capable d'énumérer les paragraphes qui le composent. Cependant, cette énumération contient également les tables du texte, aussi est-il nécessaire de vérifier que l'élément obtenu supporte le service de paragraphe. Nous allons dans l'exemple suivant supprimer entièrement les paragraphes ayant un style donné.

```
rem Code11-11.sxw   bibli : Standard Module2
Option Explicit

Sub Suppr_paragr_style()
Dim monDocument As Object
Dim listePargr As Object, elementTexte As Object
Dim uneSuppression As Boolean
monDocument = ThisComponent
Do
  uneSuppression = false
  listePargr = monDocument.Text.createEnumeration
```

```
  Do While listePargr.hasMoreElements
    elementTexte = listePargr.nextElement
    if elementTexte.supportsService("com.sun.star.text.Paragraph") Then
      if elementTexte.paraStyleName= "monStyle" then
        elementTexte.dispose
        uneSuppression = true
      end if
    end if
  Loop
Loop While uneSuppression
End Sub
```

La boucle `While` la plus externe est nécessaire pour ne pas « oublier » certains paragraphes. La fonction `createEnumeration` de l'objet texte renvoie un objet capable, d'une part de signaler l'existence d'éléments non encore vus (la fonction `hasMoreElements` qui renvoie alors `true`), et d'autre part de fournir chacun d'eux (avec la fonction `nextElement`). Pour distinguer les éléments paragraphes, nous employons la fonction `supportsService`. Le style d'un paragraphe est exposé dans la propriété `ParaStyleName`.

À l'exécution de la macro, vous constaterez que les paragraphes non concernés ont bien conservé leur style.

Appliquer un formatage

Le moyen le plus efficace de formater un texte avec une macro est de lui appliquer des styles définis au préalable. Le plus simple est de partir d'un document modèle écrit manuellement, dans lequel on a défini les styles nécessaires.

Appliquer un style à un paragraphe

Le curseur d'écriture possède une propriété appelée `ParaStyleName` qui indique le nom du style du paragraphe dans lequel le curseur se trouve. Cette propriété est une chaîne de caractères. La macro suivante affecte le style « Titre 4 » au deuxième paragraphe du document en cours.

```
rem Code11-02.sxw   bibli : Formatage Module1
Option Explicit

Sub AffecterStyleParagraphe()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur= monTexte.createTextCursor
```

```
monCurseur.gotoNextParagraph(false)
monCurseur.paraStyleName= "Titre 4"
End Sub
```

Le curseur d'écriture peut être dans une position quelconque dans le paragraphe. Inversement, la lecture de la propriété `paraStyleName` fournit le nom du style du paragraphe dans lequel se trouve le curseur.

PIÈGE Les noms de styles traduits

Pour les styles standards fournis avec OpenOffice.org, vous récupérez dans `paraStyleName` le nom anglais du style, même avec une version francisée. Dans la macro exemple, on affecte le style « Titre 4 » et on relira le style « Heading 4 ». En revanche, les styles que vous créez n'ont évidemment qu'un seul nom. L'annexe B offre une fonction `getLocaleStyleName` qui traduit un nom de style anglais dans son nom localisé.

Attention à la casse ! La chaîne de caractères du style que vous affectez doit reproduire exactement le nom du style existant : majuscules, minuscules, accents. Sinon le style du paragraphe restera inchangé.

Appliquer un style à un ou plusieurs caractères

Le curseur d'écriture possède une propriété appelée `CharStyleName`, qui indique le nom du style du caractère courant ou des caractères sélectionnés par le curseur. Cette propriété est une chaîne de caractères. La macro suivante sélectionne une zone et lui applique un style de caractères.

```
rem Code11-02.sxw   bibli : Formatage Module2
Option Explicit

Sub AffecterStyleCaractere()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur= monTexte.createTextCursor
monCurseur.gotoNextParagraph(false)
monCurseur.gotoNextWord(false)
monCurseur.gotoEndOfWord(true)
monCurseur.CharStyleName = "MonStyleCaract"
End Sub
```

La propriété `CharStyleName` peut fournir le nom du style d'un caractère sélectionné, à condition qu'il ne soit pas formaté « Par défaut ». Pour les styles de caractère standards fournis avec OpenOffice.org, vous obtiendrez le nom anglais du style, même avec une version francisée. Si vous sélectionnez plusieurs caractères de styles différents, la propriété `CharStyleName` donne une chaîne vide.

Formatage local des caractères

Ces formatages utilisent diverses propriétés de l'objet curseur. La méthode est identique dans chaque cas ; seul le premier est traité en exemple complet.

La « graisse »

En terme d'imprimerie, la « graisse » est l'épaisseur des pleins de la lettre. On utilise la propriété `CharWeight` de type `Single`, à laquelle on affecte une constante nommée de la forme :

```
com.sun.star.awt.FontWeight.NORMAL
```

Cet exemple met un mot en gras.

```
rem Code11-02.sxw   bibli : Formatage Module3
Option Explicit

Sub FormaterCaracteres()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur= monTexte.createTextCursor
monCurseur.gotoNextParagraph(false)
monCurseur.gotoNextWord(false)
monCurseur.gotoEndOfWord(true)
monCurseur.CharWeight = com.sun.star.awt.FontWeight.BOLD
End Sub
```

Les différentes valeurs de constantes possibles sont listées dans le tableau 11-5, du plus maigre au plus gras. Il s'agit de nombres réels : par exemple 150 % vaut 1,5.

Attention à la casse ! Les constantes nommées doivent être écrites en respectant les majuscules et minuscules.

Tableau 11–5 Constantes de Graisse

Constante	Signification	Proportion de la graisse normale
THIN	Fin.	50%
ULTRALIGHT	Ultra-maigre.	60%
LIGHT	Maigre.	75%
SEMILIGHT	Semi-maigre.	90%
NORMAL	Normal.	100%
SEMIBOLD	Semi-gras.	110%
BOLD	Gras.	150%
ULTRABOLD	Ultra-gras.	175%
BLACK	Noir.	200%

Italique

L'italique est réglée par la propriété `CharPosture` de type `Integer`, qui a deux valeurs possibles :

```
com.sun.star.awt.FontSlant.NONE
com.sun.star.awt.FontSlant.ITALIC
```

La valeur `NONE` donne un caractère droit, l'autre donne un caractère en italique.

Soulignement

OpenOffice.org dispose de nombreuses valeurs de soulignement, qu'on affecte à la propriété `CharUnderline`. de type `Integer`. Ce sont des constantes nommées (tableau 11-6) de la forme :

```
com.sun.star.awt.FontUnderline.SINGLE
```

Tableau 11–6 Constantes de soulignement

Constante	Résultat
NONE	Aucun soulignement.
SINGLE	Ligne unique.
DOUBLE	Ligne double.
DOTTED	Ligne pointillée.
DASH	Ligne de tirets.
LONGDASH	Ligne de tirets longs.
DASHDOT	Tiret-point.
DASHDOTDOT	Tiret-point-point.
SMALLWAVE	Petite ondulation.
WAVE	Ondulation.
DOUBLEWAVE	Double ondulation.
BOLD	Ligne grasse.
BOLDDOTTED	Ligne pointillée en gras.
BOLDDASH	Ligne de tirets gras.
BOLDLONGDASH	Ligne de tirets longs et gras.
BOLDDASHDOT	Tiret-point gras.
BOLDDASHDOTDOT	Tiret-point-point gras.
BOLDWAVE	Ondulation grasse.

Le soulignement peut prendre une autre couleur que celle du caractère au-dessus. Deux autres propriétés sont alors nécessaires :

- `CharUnderlineColor`, de type `Long`, reçoit la valeur de couleur du soulignement.
- `CharUnderlineHasColor`, de type `Boolean`, doit prendre la valeur `True` pour spécifier que le soulignement ne suit pas la couleur du caractère.

Notez aussi la propriété `CharWordMode`, de type `Boolean` : si sa valeur est `True`, les espaces ne seront pas soulignés.

Accentuation

Ce type de surlignement ou de soulignement est prévu pour les caractères asiatiques, mais peut être utile pour enjoliver certains textes. La propriété `CharEmphasis` reçoit une constante nommée de la forme :

```
com.sun.star.text.FontEmphasis.DOT_ABOVE
```

Le tableau 11-7 liste les différentes valeurs.

Tableau 11–7 Constantes d'accentuation

Constante	Signification
NONE	Pas d'accentuation.
DOT_ABOVE	Un point au-dessus du caractère.
CIRCLE_ABOVE	Un cercle au-dessus du caractère.
DISK_ABOVE	Un disque au-dessus du caractère.
ACCENT_ABOVE	Un accent au-dessus du caractère.
DOT_BELOW	Un point au-dessous du caractère.
CIRCLE_BELOW	Un cercle au-dessous du caractère.
DISK_BELOW	Un disque au-dessous du caractère.
ACCENT_BELOW	Un accent au-dessous du caractère.

Relief

La propriété `CharRelief`, de type `Integer`, dessine un effet de relief au caractère. Ceci n'est visible que sur une taille suffisamment grande. La propriété reçoit une des trois constantes nommées :

```
com.sun.star.text.FontRelief.NONE     ' aucun relief
com.sun.star.text.FontRelief.EMBOSSED ' gravé en relief
com.sun.star.text.FontRelief.ENGRAVED ' gravé en creux
```

Changement de casse

La propriété `CharCaseMap` transforme la casse du caractère ; elle utilise des constantes nommées (tableau 11-8) de la forme :

```
com.sun.star.style.CaseMap.NONE
```

Tableau 11–8 Constantes de changement de casse

Constante	Signification
NONE	La casse n'est pas modifiée.
UPPERCASE	Le caractère est mis en majuscules.
LOWERCASE	Le caractère est mis en minuscules.
TITLE	Le premier caractère de chaque mot est mis en majuscule.
SMALLCAPS	Le caractère est mis en petite minuscule.

Exposant et indice

Mettre un caractère en exposant ou en indice nécessite deux propriétés de type `Integer` :

- `CharEscapement` spécifie la position verticale du caractère par rapport à un caractère normal, en pourcentage de la hauteur de la police : une valeur positive pour un exposant, une valeur négative pour un indice.
- `CharEscapementHeight` spécifie la taille du caractère, en pourcentage, par rapport à la taille de la police. La valeur est positive.

Exemple :

```
' position : plus haut de 20% de la hauteur de la police
monCurseur.CharEscapement = 20
' taille : 70% de la taille de la police
monCurseur.CharEscapementHeight = 70
```

Astuce : les valeurs de position du caractère sont visibles avec l'interface utilisateur ; sélectionner le caractère et ouvrir le menu Format > Caractères... onglet Position.

Remarque : la documentation de l'API (version 1.1 disponible en janvier 2004) est incorrecte pour la propriété `CharEscapementHeight`.

Couleurs

Les propriétés de caractère indiquant une couleur sont listées dans le tableau 11-9. Le codage des couleurs et la fonction `RGB` sont expliqués au chapitre 8.

Exemple :

```
monCurseur.CharColor = RGB(250,0,0)
monCurseur.CharBackColor = 1234567
```

Tableau 11–9 Propriétés de couleur de caractère

Propriété	Type	Signification
CharColor	Long	Couleur de la police de caractère. La valeur -1 correspond à la couleur "Automatique".
CharBackColor	Long	Couleur du fond.
CharBackTransparent	Boolean	True si la couleur de fond n'est pas utilisée.
CharUnderlineColor	Long	Couleur de soulignement. La valeur -1 correspond à la couleur "Automatique".
CharUnderlineHasColor	Boolean	True pour imposer la couleur de soulignement. False pour que le soulignement prenne la même couleur que le caractère.

La couleur « Automatique » de la police de caractères se traduit par du noir si l'arrière-plan est clair, ou du blanc si l'arrière-plan est (très) sombre. La valeur -1 doit être appliquée directement car la fonction RGB ne peut la fournir.

Autres propriétés de caractère

Ces propriétés diverses sont listées dans le tableau 11-10.

Tableau 11–10 Autres propriétés de formatage de caractère

Propriété	Type	Signification
CharStyleName	String	Nom du style de caractère (respecter la casse).
CharFontName	String	Nom de la police de caractères (respecter la casse).
CharHeight	Single	Taille du caractère, en points, exemple : 9,7.
CharShadowed	Boolean	Valeur True pour ombrer le caractère.
CharCrossedOut	Boolean	Valeur True pour barrer le caractère.
CharContoured	Boolean	Valeur True pour dessiner un contour du caractère (visible sur une grande taille).
CharFlash	Boolean	Valeur True pour faire clignoter le caractère.
CharLocale	Object	Région utilisée pour les fonctions dépendant de la localisation. Voir explications ci-dessous.

La propriété `CharLocale` est une structure composée de trois éléments détaillés au tableau 11-11. Les deux premiers sont utilisés, par exemple, pour distinguer le français parlé en France du français parlé au Canada.

Tableau 11–11 Éléments descriptifs de localisation

Propriété	Type	Signification
`Language`	`String`	Code de langue ISO 639, sous forme de deux lettres en minuscules. Une liste en français se trouve à l'adresse http://www.termisti.refer.org/iso639.htm
`Country`	`String`	Code de pays ISO 3166, sous forme de deux lettres en majuscules. Une liste en français se trouve à l'adresse http://whois.perl-gratuit.com/documentation/codes_iso.html
`Variant`	`String`	Variante spécifique de certains logiciels ou vendeurs.

Supprimer tout formatage de caractère

Attention : les méthodes décrites ici ne fonctionnent pas correctement actuellement (versions 2.0.0 et antérieures d'OpenOffice.org). Voir le rapport d'anomalie IZ 55228.

Pour tout nettoyer, c'est-à-dire ne laisser que le formatage propre au style du paragraphe en cours, on applique la méthode `setAllPropertiesToDefault` de l'objet curseur.

```
monCurseur.setAllPropertiesToDefault
```

Pour remettre une propriété de formatage à sa valeur par défaut, utiliser la méthode `setPropertyToDefault` en précisant la propriété souhaitée :

```
monCurseur.setPropertyToDefault("CharHeight")
```

Curseur visible et zone sélectionnée par l'utilisateur

Obtenir le curseur visible

Le curseur visible, c'est la barre verticale clignotante affichée sur votre texte à l'écran. C'est aussi une zone sélectionnée par l'utilisateur, par exemple en faisant glisser la souris sur le texte.

L'objet curseur visible ne s'obtient pas directement de l'objet document, mais de l'objet contrôleur associé à la fenêtre courante du document.

```
Dim monDocument As Object
Dim CurseurVisible As Object
monDocument = ThisComponent
curseurVisible = monDocument.CurrentController.ViewCursor
```

Il n'existe qu'un seul curseur visible (alors qu'on peut créer plusieurs curseurs d'écriture). La variable curseurVisible reflète l'état réel du curseur visible : s'il évolue sous l'action de l'utilisateur, la variable pointe sur la nouvelle zone.

Zone sélectionnée par l'utilisateur

Modifier le contenu de la zone

Le curseur visible connaît la ou les zones sélectionnée(s) par l'utilisateur. Le formatage de ces zones peut être réalisé en employant directement le curseur visible comme un curseur d'écriture. Par exemple, la macro suivante, bien pratique, affecte un style de caractère aux zones sélectionnées par l'utilisateur. Il suffit ensuite de déclencher la macro par un raccourci clavier ou un nouveau bouton sur une barre d'outils.

```
Sub StyleCarPerso
Dim CurseurVisible As Object

CurseurVisible = ThisComponent.currentcontroller.ViewCursor
CurseurVisible.CharStyleName = "MonStyleCaract"
End Sub
```

Voici comment récupérer le texte sélectionné par l'utilisateur et le remplacer par un autre texte, en utilisant la propriété `String` du curseur visible :

```
rem Code11-01.sxw   bibli : SelectionTexte Module1a
Option Explicit

Sub ManipulerSelectionUtilisateur()
Dim monDocument As Object, CurseurVisible As Object
Dim texteSel As String
monDocument = ThisComponent
CurseurVisible = monDocument.currentcontroller.ViewCursor
texteSel = CurseurVisible.String
print texteSel
CurseurVisible.String = "BBBBB"
End Sub
```

Définir un curseur d'écriture sur la zone sélectionnée

Malheureusement, la variable `CurseurVisible` n'est pas un curseur d'écriture, mais un curseur d'un autre type, moins élaboré. Pour obtenir un curseur d'écriture à partir d'une

zone sélectionnée, nous allons utiliser la méthode déjà vue dans la section « Créer un curseur à partir d'un autre curseur » :

```
Dim unCurseur As Object
unCurseur = monTexte.createTextCursorByRange(CurseurVisible)
```

Nous pouvons préciser si le curseur d'écriture couvre la zone de sélection du curseur visible ou s'il est un point d'insertion en début ou fin de zone :

```
' curseur positionné sur la dernière sélection effectuée
unCurseur = monTexte.createTextCursorByRange(CurseurVisible)
' curseur au début de la dernière sélection effectuée
unCurseur = monTexte.createTextCursorByRange(CurseurVisible.Start)
' curseur à la fin de la dernière sélection effectuée
unCurseur = monTexte.createTextCursorByRange(CurseurVisible.End)
```

S'il n'y a pas de sélection, ces trois instructions donnent le même résultat. En cas de besoin, on détermine s'il y a une sélection avec la fonction `isCollapsed` de l'objet curseur :

```
unCurseur = monTexte.createTextCursorByRange(CurseurVisible)
if unCurseur.isCollapsed then
  ' ici pas de sélection
else
  ' ici il y a une sélection
end if
```

Obtenir l'objet texte de la sélection

Vous avez peut-être constaté que le curseur d'écriture obtenu ne fonctionne pas si le curseur visible est dans un tableau du document, ou un en-tête, ou un pied de page. Pour traiter le cas général, il nous faut retrouver l'objet texte dans lequel se trouve le curseur visible. Justement, cet objet nous est fourni par la propriété `Text` du curseur visible. Reprenons l'exemple précédent avec quelques modifications.

```
rem Code11-01.sxw   bibli : SelectionTexte Module1b
Option Explicit

Sub ManipulerToutesSelectionUtilisateur()
Dim monDocument As Object, CurseurVisible As Object
Dim monCurseur As Object, monTexte As Object
Dim texteSel As String
monDocument = ThisComponent
CurseurVisible = monDocument.currentcontroller.ViewCursor
monTexte = CurseurVisible.Text
monCurseur = monTexte.createTextCursorByRange(CurseurVisible)
```

```
texteSel = monCurseur.String
print texteSel
monCurseur.String = "BBBBB"
End Sub
```

Maintenant, la variable `monTexte` ne représente plus systématiquement le texte principal du document, mais une zone de texte quelconque. Nous pouvons lire et modifier une zone sélectionnée par l'utilisateur, même dans un en-tête, un bas de page, un tableau, etc.

Ce code marche correctement si l'utilisateur a sélectionné une seule zone dans le texte principal. Cependant, on peut sélectionner plusieurs zones simultanément, en utilisant la touche Ctrl avec glissement de la souris, pour les zones supplémentaires. Et dans ce cas, la propriété `String` fournit une chaîne nulle.

L'objet `CurrentSelection` de l'objet document fournit une liste des zones sélectionnées. On utilise alors une boucle pour trouver successivement ces zones. Le nombre de zones est dans la propriété `Count` de l'objet `CurrentSelection`. Pour chaque zone, on crée un curseur d'écriture suivant la méthode précédemment exposée. L'exemple suivant met en caractères gras les zones sélectionnées par l'utilisateur.

```
rem Code11-01.sxw   bibli : SelectionTexte Module2
Option Explicit

Sub BalayerSelectionsUtilisateur()
Dim monDocument As Object
Dim lesZones As Object, uneZone As Object
Dim monCurseur As Object, monTexte As Object
Dim x As Long
monDocument = ThisComponent
lesZones = monDocument.CurrentSelection
for x = 0 to lesZones.Count -1
  uneZone = lesZones(x)
  monTexte = uneZone.Text
  monCurseur = monTexte.createTextCursorByRange(uneZone)
  if not monCurseur.isCollapsed then
    monCurseur.CharWeight = com.sun.star.awt.FontWeight.BOLD
  end if
next
End Sub
```

Si vous n'êtes pas sûr que l'utilisateur n'a sélectionné qu'une seule zone, employez le codage ci-dessus. S'il n'y a aucune sélection, la boucle `for` est exécutée une fois. Le test `isCollapsed` évite de mettre en gras si la zone est ponctuelle.

Où se trouve le curseur ?

L'objet curseur (curseur visible ou curseur d'écriture) offre plusieurs propriétés qui nous donnent des informations sur son environnement ; d'abord, sur le texte lui-même, qui n'est pas toujours le texte principal. La distinction se fait par le contenu de la propriété `ImplementationName` de l'objet texte :

```
print CurseurVisible.Text.ImplementationName
```

Cette propriété, de type `String`, nous renseigne sur la variante d'objet texte :

- `SwXBodyText` : le texte principal ou le texte dans un dessin ;
- `SwXTextFrame` : le texte dans un cadre ;
- `SwXCell` : le texte dans une cellule de tableau ;
- `SwXHeadFootText` : le texte d'un en-tête ou bas de page ;
- `SwXFootnote` : le texte d'une note de bas de page ou d'une note de fin.

De plus, le curseur expose plusieurs propriétés permettant de remonter à l'objet dans lequel il se trouve :

- `TextTable` est le tableau ;
- `Cell` est la cellule dans le tableau ;
- `TextFrame` est le cadre ;
- `TextField` est le champ de texte ;
- `TextSection` est la section.

Lorsque le curseur n'est pas dans une de ces zones, la propriété correspondante n'est pas disponible. Pour éviter une erreur d'exécution, on utilise la fonction OOoBasic `IsEmpty` :

```
if IsEmpty(CurseurVisible.TextTable) then
  print "En dehors d'un tableau"
else
  MsgBox("Tableau : " & CurseurVisible.TextTable.Name & chr(13) & _
         "Cellule : " & CurseurVisible.Cell.CellName)
end if
```

Si vous avez l'intention d'analyser un document inconnu avec ces informations, n'oubliez pas que les combinaisons sont possibles, par exemple un tableau dans un cadre dans le texte d'une section elle-même incluse dans une section...

Explorer la zone sélectionnée par l'utilisateur

Nous allons reprendre le code donné précédemment, qui supprime des paragraphes. Cette nouvelle version ne supprime les paragraphes que dans la zone sélectionnée par l'utilisateur :

```
rem Code11-01.sxw   bibli : SelectionTexte Module3
Option Explicit
```

```
Sub ExplorerSelectionUtilisateur()
Dim monDocument As Object, monTexte As Object
Dim CurseurVisible As Object, monCurseur As Object
monDocument = ThisComponent
CurseurVisible = monDocument.currentcontroller.ViewCursor
monTexte = CurseurVisible.Text
' créer un curseur à la position de début du curseur visible
monCurseur = monTexte.createTextCursorByRange(CurseurVisible.Start)
Do While monCurseur.gotoNextParagraph(false)
  if monTexte.compareRegionEnds(monCurseur, CurseurVisible)< 0 then Exit Do
  monCurseur.goLeft(1, true)
  monCurseur.String = " "
Loop
End Sub
```

Notez que, si l'utilisateur sélectionne plusieurs zones, la macro ne supprime rien.

La fonction `compareRegionEnds` de la variable objet `monTexte` compare les positions de fin dans les deux curseurs. Elle renvoie :

1	si le premier curseur se termine avant le deuxième ;
0	si les deux se terminent à la même position ;
-1	si le premier curseur se termine après le deuxième.

Il existe aussi la fonction `compareRegionStarts`, qui renvoie :

1	si le premier curseur commence avant le deuxième ;
0	si les deux commencent à la même position ;
-1	si le premier curseur commence après le deuxième.

Sélectionner de manière visible une zone de texte

Après avoir sélectionné une zone en déplaçant un curseur d'écriture, vous voulez afficher cette sélection à l'utilisateur. On utilise pour cela une méthode de l'objet `CurrentController` :

```
rem Code11-01.sxw   bibli : SelectionTexte Module4
Option Explicit

Sub AfficherSelection()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextWord(false)
monCurseur.gotoEndOfWord(true) ' sélectionner un mot
monDocument.CurrentController.Select(monCurseur)
End Sub
```

La méthode `Select` utilise en argument un objet curseur, qui indique la zone de texte à afficher comme sélection. Le document exemple du zip téléchargeable contient deux pages. Si la fenêtre Writer est sur la première page, l'exécution de la macro fera dérouler le texte pour afficher la nouvelle position du curseur. Attention : le déroulement du texte affiché ne se fera que si la macro est exécutée en dehors de l'EDI.

La méthode `gotoRange` du curseur visible offre une autre possibilité, la zone cible étant indiquée par un curseur d'écriture :

```
curseurVisible.gotoRange(monCurseur, True)
```

Déplacer le curseur visible

Le curseur visible offre diverses méthodes listées dans le tableau 11-12, qui permettent de le déplacer dans le document, de manière très similaire à un curseur d'écriture.

Tableau 11–12 Méthodes de déplacement du curseur visible

Méthode	Signification	Retour
`Page`	Obtenir le numéro de la page en cours.	`Integer`
`jumpToFirstPage`	Aller à la première page.	Non significatif.
`jumpToLastPage`	Aller à la dernière page.	Non significatif.
`jumpToPage(n)`	Aller à la page numéro n (valeur `Integer`).	Non significatif.
`jumpToNextPage`	Aller à la page suivante.	`True` si l'action a pu être réalisée.
`jumpToPreviousPage`	Aller à la page précédente.	`True` si l'action a pu être réalisée.
`jumpToEndOfPage`	Aller à la fin de la page en cours.	Non significatif.
`jumpToStartOfPage`	Aller au début de la page en cours.	Non significatif.
`goRight(n,SEL)`	Déplacer de n caractères à droite.	`True` si l'action a pu être réalisée.
`goLeft(n,SEL)`	Déplacer de n caractères à gauche.	`True` si l'action a pu être réalisée.
`goDown(n,SEL)`	Déplacer de n caractères vers le bas.	`True` si l'action a pu être réalisée.
`goUp(n,SEL)`	Déplacer de n caractères vers le haut.	`True` si l'action a pu être réalisée.
`gotoStart(SEL)`	Déplacer au début du texte entier.	Pas de résultat.
`gotoEnd(SEL)`	Déplacer à la fin du texte entier.	Pas de résultat.
`gotoRange(zone,SEL)`	Positionner le curseur sur une zone donnée.	Pas de résultat.
`screenDown`	Aller à la page écran suivante.	`True` si l'action a pu être réalisée.
`screenUp`	Aller à la page écran précédente.	`True` si l'action a pu être réalisée.
`gotoEndOfLine(SEL)`	Aller en fin de ligne.	Pas de résultat.
`gotoStartOfLine(SEL)`	Aller en début de ligne.	Pas de résultat.
`isAtStartOfLine`	Le curseur est-il en début de ligne ?	`True` si oui.
`isAtEndOfLine`	Le curseur est-il en fin de ligne ?	`True` si oui.

La numérotation des pages est continue sur l'ensemble du document. Voici un exemple qui obtient le nombre total de pages du document, puis affiche la page 26. Notez que le document du zip téléchargeable ne contient que 2 pages et n'est pas significatif.

```
rem Code11-01.sxw   bibli : SelectionTexte Module5
Option Explicit

Sub NombreDePagesDuDocument()
Dim monDocument As Object
Dim CurseurVisible As Object
monDocument = ThisComponent
curseurVisible = monDocument.CurrentController.ViewCursor
curseurVisible.jumpToLastPage
print curseurVisible.Page
curseurVisible.jumpToPage(26)
End Sub
```

Définir des positions de tabulations

Insérer une tabulation dans un texte consiste à ajouter le caractère dont la valeur Unicode est 9, c'est-à-dire `chr(9)`.

Les positions de tabulations (les taquets) sont celles définies dans le style du paragraphe en cours, ou celles définies par défaut. Si vous souhaitez des taquets spécifiques, vous avez deux solutions :

1. La plus simple et pratique est d'utiliser un style de paragraphe que vous avez défini dans votre document.
2. L'autre consiste à définir laborieusement par programmation les taquets du paragraphe en cours, comme dans la macro ci-dessous.

```
rem Code11-03.sxw   bibli : Tabulations Module1
Option Explicit

Sub MettreTaquetsTabulation()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, Texte1 As String
Dim Tab As String
Dim PositionTaquet As New com.sun.star.style.TabStop
Dim ListeTaquets(2) As Object ' prévoir trois taquets
Tab = chr(9) ' caractère de tabulation

With PositionTaquet ' définition des taquets
  .DecimalChar = ","
  .FillChar = " "
  .Position = 2500 ' 25 mm ( 2,5 cm )
  .Alignment = com.sun.star.style.TabAlign.LEFT
```

```
  ListeTaquets(0) = PositionTaquet
  .Position = 4700 ' 47 mm
  .Alignment = com.sun.star.style.TabAlign.CENTER
  ListeTaquets(1) = PositionTaquet
  .Position = 7010 ' 70,1 mm
  .Alignment = com.sun.star.style.TabAlign.RIGHT
  ListeTaquets(2) = PositionTaquet
end With

monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(false) 'sauter le titre
' mettre les taquets sur le paragraphe en cours
monCurseur.ParaTabStops = ListeTaquets()
' insérer un texte avec tabulations
Texte1 = "Début" & Tab & "Tab0" & Tab & "Tab1" & Tab & "Tab2"
monTexte.insertString( monCurseur, Texte1, false)
End Sub
```

Vous devez affecter tous les taquets en une seule fois, avec un tableau comportant le nombre de taquets nécessaires. Chaque taquet est une structure composée de plusieurs valeurs indiquées au tableau 11-13.

Tableau 11–13 Composantes d'un taquet

Élément	Type	Signification
`Position`	`Long`	La position du taquet par rapport à la marge gauche, en 1/100 de mm.
`Alignment`	`Long`	Le type de tabulation, sous forme d'une constante, voir le tableau suivant.
`DecimalChar`	`String`	Le caractère séparateur décimal (en France, la virgule).
`FillChar`	`String`	Le caractère de suite, qui remplit les blancs, par exemple un pointillé.

La composante `Alignment` précise le type de tabulation. La valeur est une constante nommée (tableau 11-14) de la forme :

```
com.sun.star.style.TabAlign.CENTER
```

Tableau 11–14 Constantes d'alignement de tabulation

Constante	Position du taquet de tabulation
`LEFT`	À gauche du texte à tabuler.
`CENTER`	Au centre du texte à tabuler.
`RIGHT`	À droite du texte à tabuler.
`DECIMAL`	Sur le séparateur de décimales.
`DEFAULT`	Position suivant la valeur par défaut.

Rechercher, remplacer

Les mécanismes de recherche dans un document Writer utilisent un objet « descripteur de recherche ». Nous allons l'utiliser dans un premier exemple qui consiste à trouver dans le document toutes les occurrences d'un mot. Chaque mot trouvé est ensuite mis en exergue avec un arrière-plan coloré.

```
rem Code11-04.sxw   bibli : Rechercher Module1
Option Explicit

Sub TrouverToutPartout()
Dim monDocument As Object
Dim jeCherche As Object, trouv As Variant
Dim x As Long
monDocument = ThisComponent
jeCherche = monDocument.createSearchDescriptor
with jeCherche
  .SearchString = "Marseille"
  .SearchWords = true
end with
trouv = monDocument.findAll(jeCherche)
print "Nombre d'occurrences : " & trouv.Count
for x = 0 to trouv.Count -1
  trouv(x).CharBackColor = 1234567 ' fond vert sombre
next
End Sub
```

Un descripteur de recherche est obtenu avec la méthode `createSearchDescriptor` de l'objet document. Ce descripteur est une structure qui comporte plusieurs éléments à remplir avant de lancer la recherche. L'objet document fournit la fonction `findAll`, qui a pour argument l'objet descripteur de recherche, et renvoie un objet conteneur contenant toutes les occurrences trouvées. Leur nombre est disponible dans la propriété `Count` de l'objet conteneur. Basic permet d'obtenir les occurrences par une simple indexation du conteneur ; ce sont des zones de texte, qui peuvent servir à effectuer des modifications simples ou à créer un curseur d'écriture pour plus de possibilités.

La structure Basic `With ... End With` évite de répéter le nom de la variable descripteur sur plusieurs lignes. Sans elle, on écrirait :

```
jeCherche.SearchString = "Marseille"
jeCherche.SearchWords = true
```

Le descripteur de recherche

Le tableau 11-15 liste les éléments du descripteur ; leur utilisation est identique à la fonction de recherche de l'interface utilisateur :

Tableau 11–15 Descripteur de recherche

Élément	Type	Signification
`SearchString`	`String`	La chaîne de caractères à rechercher.
`SearchBackwards`	`Boolean`	`True` pour faire une recherche à reculons (ça peut servir) ; par défaut, on recherche dans le sens normal de lecture.
`SearchCaseSensitive`	`Boolean`	`True` pour distinguer les majuscules des minuscules dans la recherche ; par défaut, il n'y a pas de distinction. Quelle que soit la valeur de cette propriété, les caractères accentués sont toujours différenciés des caractères non accentués.
`SearchWords`	`Boolean`	`True` pour ne rechercher que des mots ; par défaut, on recherche la séquence de caractères n'importe où.
`SearchRegularExpression`	`Boolean`	`True` pour faire une recherche avec la méthode des expressions régulières ; par défaut, on recherche une simple égalité de chaîne.
`SearchStyles`	`Boolean`	`True` pour rechercher des paragraphes d'un style donné par `SearchString` ; par défaut, on cherche du texte.
`SearchSimilarity`	`Boolean`	`True` pour rechercher un texte similaire au texte cherché.
`SearchSimilarityRelax`	`Boolean`	`True` pour essayer toute combinaison des trois critères suivants qui permette de retrouver le texte cherché.
`SearchSimilarityRemove`	`Integer`	Nombre de caractères à retrancher pour retrouver le texte cherché.
`SearchSimilarityAdd`	`Integer`	Nombre de caractères à ajouter pour retrouver le texte cherché.
`SearchSimilarityExchange`	`Integer`	Nombre de caractères à changer pour retrouver le texte cherché.

> **PIÈGE Nom du style recherché**
>
> Pour rechercher un style, la propriété `SearchString` doit contenir le nom localisé du style, et non pas le nom anglais. Relisez à ce sujet la section « Appliquer un style à un paragraphe ».

Limiter le champ de la recherche

Si vous exécutez l'exemple précédent sur le document fourni dans le zip téléchargeable, vous constaterez des effets gênants : le document comporte un en-tête, un bas de page, un tableau et, comme par hasard, ces zones contiennent aussi le mot recherché. Elles sont toutes modifiées par la macro, alors qu'en réalité seul le texte principal nous intéressait.

Pour éviter ce désastre, nous allons vérifier que chaque zone de texte trouvée est bien dans le texte principal. La fonction Basic `EqualUnoObjects` sera utilisée pour vérifier que la propriété `Text` de la zone trouvée et la propriété `Text` du document indiquent bien le même objet.

```
rem Code11-04.sxw   bibli : Rechercher Module2
Option Explicit
```

```
Sub TrouverToutdansleTexte()
Dim monDocument As Object, monTexte As Object
Dim jeCherche As Object, trouv As Variant
Dim x As Long, posTrouve As Object
monDocument = ThisComponent
monTexte = monDocument.Text ' objet = texte principal
jeCherche = monDocument.createSearchDescriptor
with jeCherche
  .SearchString = "Marseille"
  .SearchWords = true
end with
trouv = monDocument.findAll(jeCherche)
print "Nombre d'occurrences : " & trouv.Count
for x = 0 to trouv.Count -1
  posTrouve = trouv(x)
  if EqualUnoObjects(monTexte, posTrouve.Text) then
    ' L'occurrence est bien dans le texte principal
    posTrouve.CharBackColor = 1234567 ' fond vert sombre
  end if
next
End Sub
```

Toutes les occurrences sont trouvées, mais seules celles du texte principal sont utilisées.

Un autre moyen de recherche consiste à trouver la première occurrence avec la méthode `findFirst` de l'objet document, puis à rechercher les occurrences suivantes avec la méthode `findNext` jusqu'à échec de la recherche. Voici l'équivalent de l'exemple précédent, en utilisant cette technique.

```
rem Code11-04.sxw   bibli : Rechercher Module3
Option Explicit

Sub RechercherPartoutdansleTexte()
Dim monDocument As Object, monTexte As Object
Dim jeCherche As Object, posTrouve As Object
monDocument = ThisComponent
monTexte = monDocument.Text
jeCherche = monDocument.createSearchDescriptor
with jeCherche
  .SearchString = "Marseille"
  .SearchWords = true
end with
posTrouve = monDocument.findFirst(jeCherche)
Do Until isNull(posTrouve)
  if EqualUnoObjects(monTexte, posTrouve.Text) then
    ' L'occurrence est bien dans le texte principal
    posTrouve.CharBackColor = 1234567 ' fond vert sombre
  end if
posTrouve = monDocument.findNext(posTrouve.End, jeCherche)
Loop
End Sub
```

Si la recherche ne trouve rien, l'objet `posTrouve` reçoit la valeur `Null` et la boucle n'est pas exécutée.

Il est souvent nécessaire de restreindre la recherche à une certaine zone de texte dans le document. Nous allons utiliser un curseur d'écriture pour sélectionner la zone de recherche dans le texte, puis nous allons vérifier si la fin de la zone trouvée est encore dans la zone de recherche. La recherche débutera, non pas avec `findFirst`, mais avec `findNext` qui nous permet d'imposer le point de départ de la recherche.

```
rem Code11-04.sxw   bibli : Rechercher Module4
Option Explicit

Sub RechercherDansPartieDeTexte()
Dim monDocument As Object, monTexte As Object
Dim jeCherche As Object, posTrouve As Object
Dim monCurseur As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur= monTexte.createTextCursor
monCurseur.gotoNextParagraph(false) ' début du 2ème paragr
monCurseur.gotoNextParagraph(true) ' sélection du 2ème paragr
jeCherche = monDocument.createSearchDescriptor
with jeCherche
  .SearchString = "et"
  .SearchWords = true
end with
posTrouve = monDocument.findNext(monCurseur.Start, jeCherche)
Do Until isNull(posTrouve)
  if EqualUnoObjects(monTexte, posTrouve.Text) then
    if monTexte.compareRegionEnds(posTrouve, monCurseur)< 0 then Exit Do
    ' L'occurrence est dans la zone de recherche
    posTrouve.CharBackColor = 1234567 ' fond vert sombre
  end if
  posTrouve = monDocument.findNext(posTrouve.End, jeCherche)
Loop
End Sub
```

La fonction `compareRegionEnds` renvoie -1 si la zone en premier argument commence après la zone en deuxième argument. Dans ce cas, `Exit Do` termine immédiatement la boucle. La zone de recherche pourrait être une zone sélectionnée par l'utilisateur ; ceci est décrit plus haut.

Il est possible de sélectionner visuellement la zone trouvée, au lieu de changer la couleur de fond :

```
monDocument.CurrentController.Select(posTrouve)
```

Rechercher pour remplacer

Lors d'une recherche, la zone de texte `posTrouve` sélectionne le texte trouvé. Nous pouvons utiliser cette zone de texte pour créer un curseur d'écriture et modifier le texte à notre guise. En reprenant l'exemple précédent, nous allons maintenant remplacer les mots « et » de la zone sélectionnée par le caractère « & ». De plus, si la zone trouvée est « Et », nous changerons la taille de caractère.

```
rem Code11-04.sxw   bibli : Rechercher Module5
Option Explicit

Sub RemplacerDansPartieDeTexte()
Dim monDocument As Object, monTexte As Object
Dim jeCherche As Object, posTrouve As Object
Dim monCurseur As Object, Curseur2 As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur= monTexte.createTextCursor
monCurseur.gotoNextParagraph(false) 'début du 2ème paragr
monCurseur.gotoNextParagraph(true) ' sélection du 2ème paragr
jeCherche = monDocument.createSearchDescriptor
with jeCherche
  .SearchString = "et"
  .SearchWords = true
end with
posTrouve = monDocument.findNext(monCurseur.Start, jeCherche)
Do Until isNull(posTrouve)
  if EqualUnoObjects(monTexte, posTrouve.Text) then
    if monTexte.compareRegionEnds(posTrouve, monCurseur)< 0 then Exit Do
    ' L'occurrence est dans la zone de recherche
    Curseur2 = monTexte.createTextCursorByRange(posTrouve)
    ' Curseur2 sélectionne le texte trouvé
    if Curseur2.String = "Et" then Curseur2.CharHeight = 16
    Curseur2.String = "&" ' changer le texte
  end if
  posTrouve = monDocument.findNext(posTrouve.End, jeCherche)
Loop
End Sub
```

La propriété `String` de l'objet `Curseur2` contient initialement le texte trouvé. En changeant sa valeur, nous remplaçons le texte existant par un autre. Nous aurions aussi bien pu déplacer le curseur et effectuer toute sorte de remplacement, insertion, suppression sur les textes trouvés.

Tout remplacer

Dans le cas simple où toutes les occurrences doivent être systématiquement remplacées partout, texte principal et autres textes, l'objet document offre la fonction `replaceAll`, qui effectue ce travail et renvoie le nombre de remplacements. Elle utilise un descripteur de remplacement qui contient tous les éléments du descripteur de recherche et quelques autres supplémentaires listés dans le tableau 11-16.

Tableau 11–16 Descripteur de remplacement (éléments spécifiques)

Élément	Type	Signification
`ReplaceString`	`String`	La chaîne de caractères à mettre à la place de celle trouvée.
`ValueSearch`	`Boolean`	`True` si on recherche certaines valeurs des attributs. `False` si on recherche seulement si certains attributs existent
`SearchAttributes`	`Array()`	Tableau d'attributs à rechercher (voir explications).
`ReplaceAttributes`	`Array()`	Tableau d'attributs de remplacement (voir explications).

On utilise une nouvelle méthode `createReplaceDescriptor` pour obtenir un descripteur de remplacement. Pour un simple remplacement de texte, il suffit de remplir en plus `ReplaceString`.

```
rem Code11-04.sxw   bibli : Remplacer Module1
Option Explicit

Sub RemplacerTextePartout()
Dim monDocument As Object
Dim jeCherche As Object, nbrFois As Long
monDocument = ThisComponent
jeCherche = monDocument.createReplaceDescriptor
with jeCherche
  .SearchString = "Marseille"
  .ReplaceString = "Bordeaux"
  .SearchWords = true
end with
nbrFois = monDocument.replaceAll(jeCherche)
print "Nombre de remplacements : " & nbrFois
End Sub
```

Rappelons que `replaceAll` ne permet pas de distinguer le texte principal des autres textes du document.

La macro suivante modifie tous les paragraphes de style `Style1`, pour leur affecter un autre style `Style2`. Si `Style1` est spécifique au type de texte considéré dans le document (par exemple utilisé seulement dans le texte principal), il n'y aura pas d'effet pervers.

Attention, utilisez le nom anglais du style s'il est prédéfini par OpenOffice.org (voir la section sur les styles).

```
rem Code11-04.sxw   bibli : Remplacer Module2
Option Explicit

Sub RemplacerStylePartout()
Dim MonDocument As Object
Dim JeCherche As Object
Dim MonCurseur As Object
MonDocument = ThisComponent
JeCherche = MonDocument.createReplaceDescriptor
with JeCherche
  .SearchString = "Style1"
  .ReplaceString = "Style2"
  .SearchStyles = true
end with
MonDocument.replaceAll(JeCherche)
End Sub
```

La propriété `SearchAttributes` sert à limiter la recherche à certains formatages. La propriété `ReplaceAttributes` précise les formatages à appliquer sur les zones trouvées. Il est possible d'utiliser l'une ou l'autre de ces propriétés, les deux, ou aucune. Pour remplir chacune de ces propriétés, il faut constituer un tableau de `PropertyValue` et l'affecter à la propriété souhaitée.

Dans l'exemple suivant, nous recherchons les mots « ses » qui sont soulignés d'une ondulation. Nous allons conserver le texte, mais lui imposer le formatage : sans ondulation, italique, gras et avec un fond vert. Le texte du document (voir zip téléchargeable) contient plusieurs mots « ses », mais un seul souligné en ondulé.

```
rem Code11-04.sxw   bibli : Remplacer Module3
Option Explicit

Sub RemplacerAttributs()
Dim MonDocument As Object
Dim JeCherche As Object
Dim MonCurseur As Object
Dim attRech(0) As New com.sun.star.beans.PropertyValue
Dim attRempl(3) As New com.sun.star.beans.PropertyValue

attRech(0).Name = "CharUnderline"
attRech(0).Value = com.sun.star.awt.FontUnderline.WAVE

attRempl(0).Name = "CharPosture"
attRempl(0).Value = com.sun.star.awt.FontSlant.ITALIC
attRempl(1).Name = "CharUnderline"
attRempl(1).Value = com.sun.star.awt.FontUnderline.NONE
attRempl(2).Name = "CharWeight"
```

```
attRempl(2).Value = com.sun.star.awt.FontWeight.BOLD
attRempl(3).Name = "CharBackColor"
attRempl(3).Value = 1234567 ' fond vert sombre

MonDocument = ThisComponent
JeCherche = MonDocument.createReplaceDescriptor
with JeCherche
  .SearchString = "ses"
  .ReplaceString = "ses" ' on ne change pas le texte
  .SearchWords = true
  .ValueSearch = True
  .SearchAttributes = attRech()
  .ReplaceAttributes = attRempl()
end with
MonDocument.replaceAll(JeCherche)
End Sub
```

Si on donne à la propriété `ValueSearch` la valeur `False`, tout formatage de soulignement est recherché, la valeur est ignorée. Dans ce cas, le remplacement d'attributs ne fonctionne pas, il est seulement possible de modifier le texte. Vous pouvez faire l'essai en modifiant deux lignes sur l'exemple précédent :

```
  .ReplaceString = "mes" ' on change le texte
  .ValueSearch = False ' recherche de l'existence d'attributs
```

Rechercher des paragraphes

Il est facile de parcourir un texte paragraphe par paragraphe. Dans cet extrait de code, nous recherchons dans un texte les paragraphes d'un style donné.

```
Dim listeParagr As Object, elmt As Object

listeParagr = leDocument.Text.createEnumeration
Do While listeParagr.hasMoreElements
  elmt = listeParagr.nextElement
  if elmt.supportsService("com.sun.star.text.Paragraph") Then
    Select Case elmt.paraStyleName
    Case "TitreNiveau1"
      ' faire quelque chose
    Case "TitreNiveau2"
      print elmt.String ' visualiser le contenu du paragraphe
    Case "TitreNiveau3"
      ' faire autre chose
    end select
  end if
Loop
```

Nous obtenons de l'objet texte un énumérateur de paragraphes. Les tableaux étant compris dans cette liste, nous les éliminons en vérifiant si l'élément trouvé supporte le service de paragraphe.

Attention à la casse ! Les noms de services doivent être écrits en respectant les majuscules et minuscules.

La propriété `paraStyleName` nous donne le style du paragraphe, que nous pouvons modifier par simple affectation. La propriété `String` d'un paragraphe renvoie le texte qu'il contient (avec un petit risque qu'il soit tronqué). On peut ensuite affecter un autre texte à cette propriété.

Les tableaux

Créer et manipuler un tableau Writer entièrement par programmation n'est pas une mince affaire, vous allez le voir. Ici encore, vous avez intérêt à partir d'un document existant, avec le ou les tableau(x) nécessaire(s), formaté(s) manuellement à votre convenance. Avec une macro, il suffit alors de retrouver chaque tableau grâce à son nom et de le remplir.

Insérer un tableau

On utilise un curseur pour désigner l'endroit d'insertion du tableau. La méthode `createInstance` de l'objet document fournit l'objet tableau. Il est ainsi répertorié dans l'ensemble des tableaux du document. La méthode `initialize` de l'objet tableau spécifie le nombre de lignes et de colonnes. L'insertion effective du tableau sera faite par la méthode `insertTextContent` de l'objet texte.

```
rem Code11-05.sxw   bibli : AjoutTableau Module1
Option Explicit

Sub InsererUnTableau()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, maTable As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
maTable = monDocument.createInstance("com.sun.star.text.TextTable")
maTable.initialize(5,9) ' nombre de : lignes, colonnes
monTexte.insertTextContent(monCurseur, maTable, false)
End Sub
```

L'argument de `createInstance` doit être écrit en respectant la casse.

Le troisième argument de la méthode `insertTextContent` signifie :

- `False` = insérer dans le texte ;
- `True` = écraser la zone actuellement sélectionnée par le curseur.

Insérer plusieurs tableaux

À chaque insertion d'un tableau, il est nécessaire d'obtenir un nouvel objet `TextTable`, comme dans cet exemple.

```
rem Code11-05.sxw   bibli : AjoutTableau Module2
Option Explicit

Sub InsererPlusieursTableaux()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, maTable As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
maTable = monDocument.createInstance("com.sun.star.text.TextTable")
maTable.initialize(5,9) ' nombre de : lignes, colonnes
monTexte.insertTextContent(monCurseur, maTable, false)
' - - - ( remplir le premier tableau ) - - -
monCurseur.gotoNextParagraph(False) ' aller ailleurs
maTable = monDocument.createInstance("com.sun.star.text.TextTable")
maTable.initialize(2,6) ' nombre de : lignes, colonnes
monTexte.insertTextContent(monCurseur, maTable, false)
End Sub
```

Trouver un tableau existant

Rappelons que tout tableau possède un nom (par défaut, en version française : `Tableau1`, `Tableau2`, etc. par ordre de création). Avec le Navigateur, on peut donner un nom plus spécifique à chaque tableau.

Supposons que dans un document existant nous ayons à modifier le contenu d'un tableau nommé « Finances ». L'objet document fournit la collection de ses tableaux avec la propriété `TextTables` (notez le « s » du pluriel) et on obtient un tableau d'un nom particulier avec la fonction `getByName` de l'objet collection. Avec OOoBasic, on accède à chaque tableau soit par un index, qui est un numéro d'ordre, soit de préférence par `getByName`.

```
maTable = moDocument.TextTables(2) ' accès par index
' accès par le nom
maTable = monDocument.TextTables.getByName("Finances")
```

Le `getByName` déclenchera une exception s'il n'existe aucun tableau de ce nom. Vous pouvez vérifier qu'un tel nom de tableau existe bien :

```
if monDocument.TextTables.hasByName("Finances") then
  ' le tableau existe, le modifier
end if
```

Nous pouvons maintenant modifier le tableau en utilisant la variable `maTable`.

Notez qu'il est facile de renommer un tableau :

```
maTable = monDocument.TextTables.getByName("Finances")
maTable.Name = "Finances_2003"
```

Un nom de tableau ne doit pas comporter de caractère espace.

L'objet `TextTables` nous permet de connaître tous les tableaux du document et donc de les modifier. Le nombre de tableaux est fourni par la propriété `Count`, leurs noms sont obtenus avec la propriété `Name` de chaque objet tableau.

```
rem Code11-05.sxw   bibli : AjoutTableau Module3
Option Explicit

Sub ListerLesTableaux()
Dim monDocument As Object
Dim lesTables As Object, maTable As Object
Dim x As Long
monDocument = ThisComponent
lesTables = monDocument.TextTables
for x = 0 to lesTables.Count -1
  maTable = lesTables(x)
  print "Tableau : " & maTable.Name
next
End Sub
```

Attention : l'ordre des tableaux dans la liste de `TextTables` ne correspond pas forcément à l'ordre dans le document.

Supprimer un tableau

Avec une variable `maTable` pointant sur un tableau, on supprime celui-ci en utilisant la méthode `removeTextContent` de l'objet texte.

```
if MsgBox("Effacer ce tableau ?", 132) = 6 then
  monTexte.removeTextContent(maTable)
end if
```

Propriétés de tableau

Le tableau 11-17 liste les principales propriétés de tableau.

Pour un nouveau tableau, ces propriétés peuvent être modifiées avant ou après l'instruction `initialize`, mais toujours après `createInstance`.

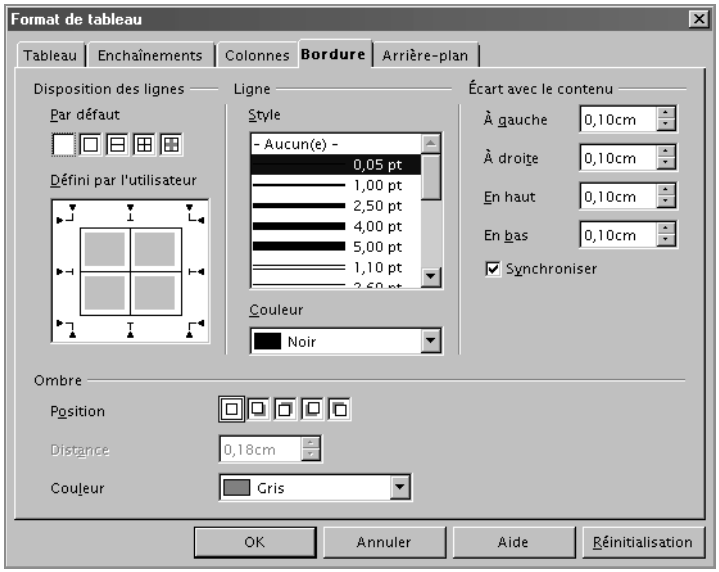
Tableau 11–17 Propriétés de tableau

Propriété	Type	Signification
`Name`	`String`	Nom du tableau.
`BreakType`	`Integer`	Impose un saut de page ou de colonne avant ou après le tableau. Mêmes valeurs que pour le texte principal.
`KeepTogether`	`Boolean`	`True` pour maintenir ce tableau et le paragraphe suivant sur la même page ou la même colonne.
`Split`	`Boolean`	`False` : impose que le tableau ne soit pas à cheval sur deux pages ou deux colonnes (à condition que le tableau tienne dans la page !) `True` (valeur par défaut) : le tableau peut s'étendre sur deux pages ou plus.
`PageDescName`	`String`	Si cette propriété contient un nom de style de page, un saut de page est effectué avant le tableau et la nouvelle page prend ce style.
`PageNumberOffset`	`Integer`	Premier numéro de page s'il y a eu un saut de page forcé.
`RepeatHeadline`	`Boolean`	`True` pour répéter la ligne d'en-tête à chaque nouvelle page. `False` par défaut.
`TopMargin`	`Long`	Marge entre le haut du tableau et le paragraphe précédent, en 1/100 de mm.
`BottomMargin`	`Long`	Marge entre le bas du tableau et le paragraphe suivant, en 1/100 de mm.
`ShadowFormat`	`Object`	Ombre du tableau (voir explications).
`BackTransparent`	`Boolean`	`True` rend le fond transparent (interface utilisateur : sans remplissage).
`BackColor`	`Long`	Couleur du fond.
`TableColumnSeparators`	`Object`	Accès aux séparateurs de colonnes du tableau, décrit plus loin.
`TableColumnRelativeSum`	`Integer`	Somme des largeurs de colonnes indiquées dans `TableColumnSeparators`. Lecture seulement.
`TableBorder`	`Object`	Bordures du tableau (voir explications).

Bordures de tableau

Référez-vous au panneau Bordures présenté par l'interface utilisateur (figure 11-1). Sur ce panneau, les largeurs de trait sont indiquées en point typographique « pica » ; un point vaut 35,28 centièmes de millimètres.

Figure 11–1
Le panneau utilisateur Bordures



Les bordures de tableau sont exposées par sa propriété `TableBorder`. Cette dernière est une structure (voir tableau 11-18) dont chaque descripteur de ligne (`TopLine`, `LeftLine`, etc.) est lui-même une structure décrite au tableau 11-19.

Tableau 11–18 Structure de TableBorder

Élément	Type	Signification
`IsTopLineValid`	`Boolean`	`True` si la bordure du haut existe.
`TopLine`	`Object`	Description de la bordure du haut.
`IsBottomLineValid`	`Boolean`	`True` si la bordure du bas existe.
`BottomLine`	`Object`	Description de la bordure du bas.
`IsLeftLineValid`	`Boolean`	`True` si la bordure de gauche existe.
`LeftLine`	`Object`	Description de la bordure de gauche.
`IsRightLineValid`	`Boolean`	`True` si la bordure de droite existe.
`RightLine`	`Object`	Description de la bordure de droite.
`IsHorizontalLineValid`	`Boolean`	`True` si la bordure des lignes horizontales existe.
`HorizontalLine`	`Object`	Description de la bordure des lignes horizontales.

Tableau 11–18 Structure de TableBorder

Élément	Type	Signification
IsVerticalLineValid	Boolean	True si la bordure des lignes verticales existe.
VerticalLine	Object	Description de la bordure des lignes verticales.
IsDistanceValid	Boolean	True si l'écart avec le contenu est utilisé.
Distance	Integer	Écart avec le contenu, en 1/100 de mm, pour toutes les bordures.

Le réglage d'un écart avec le contenu différent suivant les bordures ne semble ni réalisable ni visible à travers l'API. La propriété `Distance` correspond à l'écart « synchronisé » du panneau d'interface utilisateur.

Tableau 11–19 Structure d'une bordure

Élément	Type	Signification
Color	Long	Couleur de la ligne.
InnerLineWidth	Integer	Épaisseur de la ligne interne, en 1/100 de mm, dans le cas d'une bordure double. La valeur zéro correspond à une bordure simple.
OuterLineWidth	Integer	Épaisseur de la ligne simple, ou de la ligne externe dans le cas d'une bordure double ; en 1/100 de mm. La valeur zéro correspond à une bordure inexistante.
LineDistance	Integer	Distance entre les deux lignes d'une bordure double, en 1/100 de mm.

Modifier une bordure nécessite d'utiliser des variables intermédiaires pour accéder aux structures :

```
Dim unBord As New com.sun.star.table.BorderLine

lesBords = maTable.TableBorder
unBord.OuterLineWidth = 250 ' 2,5 mm
unBord.Color = RGB(200,0,0) ' rouge
lesBords.LeftLine= unBord ' changer la bordure gauche
maTable.TableBorder = lesBords
```

Les propriétés `IsHorizontalLineValid` et `IsVerticalLineValid` ont une signification particulière, car elles concernent le quadrillage intérieur.

- En lecture, la valeur `True` indique que toutes les lignes de ce type ont les mêmes valeurs de bordure.
- En écriture, la valeur `True` indique que toutes les lignes de ce type doivent prendre la même valeur de bordure.

Ombre de tableau

L'objet `ShadowFormat` est une structure comportant plusieurs éléments, voir le tableau 11-20.

Tableau 11–20 Structure de ShadowFormat

Élément	Type	Signification
`Location`	`Integer`	Position de l'ombre, sous forme de constante nommée, voir tableau suivant.
`ShadowWidth`	`Integer`	Largeur de l'ombre ; une valeur de 200 correspond à environ 3,5 mm.
`IsTransparent`	`Boolean`	`True` si l'ombre est transparente.
`Color`	`Long`	Couleur de l'ombre.

La position de l'ombre est exprimée sous forme de constante nommée (tableau 11-21), par exemple :

```
com.sun.star.table.ShadowLocation.BOTTOM_RIGHT
```

Tableau 11–21 Constantes de position d'ombre de tableau

Constante	Signification
NONE	Aucune ombre.
TOP_LEFT	Ombre portée vers le haut et à gauche.
TOP_RIGHT	Ombre portée vers le haut et à droite.
BOTTOM_LEFT	Ombre portée vers le bas et à gauche.
BOTTOM_RIGHT	Ombre portée vers le bas et à droite.

Il est nécessaire de remplir la structure dans une variable de travail qui servira à initialiser `ShadowFormat`. Voici un petit exemple pour clarifier, qui modifie un tableau existant du nom de « Finances » :

```
rem Code11-05.sxw   bibli : ConfigTableau Module1
Option Explicit

Sub MettreUneOmbre()
Dim monDocument As Object, maTable As Object
Dim ombre As New com.sun.star.table.ShadowFormat
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Finances")
ombre.Location = com.sun.star.table.ShadowLocation.TOP_LEFT
ombre.ShadowWidth = 200 ' environ 3,5 mm
ombre.Color = RGB(100,100,100) ' couleur grise
maTable.ShadowFormat = ombre
End Sub
```

Largeur du tableau

Il existe plusieurs propriétés inter-dépendantes qui modifient la largeur d'un tableau.

Positionnement horizontal

La propriété `HoriOrient`, du type `Integer`, définit comment le tableau est positionné horizontalement, par rapport aux marges. Elle contient une constante nommée dont les valeurs possibles sont indiquées dans le tableau 11-22. La valeur par défaut est :

```
com.sun.star.text.HoriOrientation.FULL
```

Comme la valeur par défaut est `FULL`, il est absolument nécessaire de mettre une autre valeur (autre que `NONE`) pour pouvoir imposer une largeur.

La largeur se règle avec différentes propriétés indiquées dans le tableau 11-23.

Tableau 11–22 Constantes de positionnement horizontal de tableau

Constante	Signification
`NONE`	Étaler le tableau de la marge de gauche à la marge de droite.
`RIGHT`	Aligner le tableau sur la marge de droite.
`LEFT`	Aligner le tableau sur la marge de gauche.
`CENTER`	Centrer le tableau par rapport à l'espace disponible.
`FULL`	Étaler le tableau sur tout l'espace disponible, sans marges de table.
`LEFT_AND_WIDTH`	Aligner le tableau sur la marge de gauche, avec une largeur imposée par la propriété `WIDTH`

Tableau 11–23 Propriétés de réglage de largeur de tableau

Propriété	Type	Signification
`IsWidthRelative`	`Boolean`	`True` : prendre en compte `RelativeWidth`. `False` : prendre en compte `Width`.
`RelativeWidth`	`Integer`	Valeur entre 1 et 100 : largeur en pourcentage de l'espace libre entre marges de gauche et de droite.
`Width`	`Long`	Largeur absolue du tableau. L'unité de mesure n'est pas disponible.
`LeftMargin`	`Long`	Largeur absolue entre la marge gauche et le tableau, en 1/100 de mm.
`RightMargin`	`Long`	Largeur absolue entre la marge droite et le tableau, en 1/100 de mm.

Applications pratiques

Dans ces extraits de code, nous indiquons seulement les instructions concernant la largeur. Les résultats ont été trouvés sur la version 1.1.0 d'OpenOffice.org. Un exemple complet est disponible sur le zip téléchargeable, fichier `Code11-05.sxw`, bibliothèque `ConfigTableau`, module `Module2`.

Les largeurs absolues dépendent de diverses conditions (selon la documentation de l'API), aussi faites des essais avec vos documents pour déterminer l'échelle.

1. Tableau centré, largeur 80 %

```
maTable.HoriOrient =_
  com.sun.star.text.HoriOrientation.CENTER
maTable.IsWidthRelative = true
maTable.RelativeWidth = 80
```

   La largeur de la dernière colonne est incorrecte. Les propriétés `LeftMargin` et `RightMargin` sont ignorées.

2. Tableau centré, largeur absolue

```
maTable.HoriOrient =_
  com.sun.star.text.HoriOrientation.CENTER
maTable.IsWidthRelative = false
maTable.Width = 8000
```

   La largeur obtenue est 141,1 mm. Les propriétés `LeftMargin` et `RightMargin` sont ignorées.

3. Tableau aligné à gauche, largeur 80 %

```
maTable.HoriOrient =_
  com.sun.star.text.HoriOrientation.LEFT_AND_WIDTH
maTable.IsWidthRelative = true
maTable.RelativeWidth = 80
maTable.LeftMargin = 2000 ' 20 mm
```

   La largeur de la dernière colonne est incorrecte. La propriété `RightMargin` est ignorée.

4. Tableau aligné à gauche, largeur absolue

```
maTable.HoriOrient = _
  com.sun.star.text.HoriOrientation.LEFT_AND_WIDTH
maTable.IsWidthRelative = false
maTable.Width = 4000
maTable.LeftMargin = 1000 ' 10 mm
```

   La largeur obtenue est 70,6 mm. La propriété `RightMargin` est ignorée en orientation `LEFT_AND_WIDTH`.

5 Tableau étalé entre les marges gauche et droite

```
maTable.HoriOrient =_
  com.sun.star.text.HoriOrientation.NONE
maTable.LeftMargin = 2000 ' 20 mm
maTable.RightMargin = 3500 ' 15 mm
```

Les propriétés `IsWidthRelative`, `RelativeWidth`, `Width`, sont ignorées.

Lignes

Certains objets ou propriétés décrits ici ne sont utilisables que sur un tableau constitué d'un quadrillage simple.

L'objet tableau fournit l'ensemble de ses lignes dans la propriété `Rows` , qui est un tableau (`Array`). On accède à chaque ligne par son numéro d'ordre (zéro pour la première ligne) utilisé comme un index. Le nombre de lignes du tableau est la propriété `Count` de l'objet `Rows` du tableau.

Les principales propriétés d'un objet ligne sont dans le tableau 11-24.

Tableau 11–24 Propriétés d'une ligne de tableau

Propriété	Type	Signification
`IsAutoHeight`	`Boolean`	`True` : la hauteur est adaptée au contenu de la ligne (valeur par défaut). `False` : la hauteur dépend de la propriété `Height`.
`Height`	`Long`	Hauteur de la ligne, en 1/100 de mm.
`BackColor`	`Long`	Couleur du fond.
`BackTransparent`	`Long`	`True` rend le fond transparent (interface utilisateur : sans remplissage).
`TableColumnSeparators`	`Object`	Accès aux séparateurs de colonnes de la ligne (voir description des colonnes).

Exemple :

```
rem Code11-05.sxw   bibli : LigneColonne Module1
Option Explicit

Sub ModifierLigne()
Dim monDocument As Object, maTable As Object
Dim lesLignes As Object, maLigne As Object
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Finances")
lesLignes = maTable.Rows
print "Nombre de lignes", lesLignes.Count
maLigne= lesLignes(2) ' troisième ligne
```

```
With maLigne
  .Height = 1000 ' 10 mm
  .IsAutoHeight = False ' prendre en compte Height
  .BackColor = RGB(240, 240, 0) ' couleur jaune
End With
End Sub
```

Les lignes sont numérotées à partir de zéro. Nous allons ajouter 3 lignes adjacentes, dont la première aura le rang 2, en utilisant la méthode `insertByIndex` de l'objet `Rows`.

```
rem Code11-05.sxw   bibli : LigneColonne Module2
Option Explicit

Sub InsererLignes()
Dim monDocument As Object, maTable As Object
Dim lesLignes As Object, maLigne As Object
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Finances")
lesLignes = maTable.Rows
lesLignes.insertByIndex(2,3) ' ajouter 3 lignes en position 2
print "Maintenant on va supprimer"
lesLignes.removeByIndex(2,3) ' supprimer ces trois lignes
End Sub
```

L'ancienne ligne de rang 2 est maintenant la ligne de rang 5. Nous avons ensuite utilisé une méthode symétrique, `removeByIndex`, dont le premier argument est le rang de la première ligne à supprimer et le deuxième argument est le nombre de lignes à supprimer.

Colonnes

Certains objets ou propriétés décrits ici ne sont utilisables que sur un tableau constitué d'un quadrillage simple.

L'objet tableau fournit l'ensemble de ses lignes dans la propriété `Columns` , qui est un tableau (`Array`). On accède à chaque colonne par son numéro d'ordre (zéro pour la première colonne) utilisé comme un index, mais l'objet obtenu n'a aucune propriété. Si vous souhaitez modifier toute une colonne, il faudra accéder à chacune de ses cellules. Le nombre de colonnes du tableau est la propriété `Count` de l'objet `Columns` du tableau.

L'objet `TableColumnSeparators` est défini pour un tableau si toutes les lignes ont la même structure. Il en existe aussi un par ligne de tableau. Cet objet liste les séparateurs de colonnes ; chacun est une structure composée de deux éléments, selon le tableau 11-25. Si le tableau possède 4 colonnes, il y a 3 séparateurs, de rang 0 à 2.

Tableau 11–25 Structure de TableColumnSeparators

Élément	Type	Signification
Position	Integer	Position du séparateur, par rapport au côté gauche du tableau.
IsVisible	Boolean	True si le séparateur est visible.

La propriété TableColumnRelativeSum, qui existe pour le tableau et pour une ligne, est initialisée à une certaine valeur qui représente la largeur totale (ce n'est pas une dimension). Les positions ont une valeur inférieure à celle-là ; en clair, une position est caractérisée comme une proportion de la valeur maximale possible. En changeant la position d'un séparateur, on modifie la largeur des deux colonnes adjacentes. Notez qu'on ne peut déplacer un séparateur non visible.

Dans cet exemple, nous affichons les valeurs de ces propriétés pour le tableau Résultats, puis nous modifions la position de deux séparateurs. La macro montre comment définir une position de séparateur comme une fraction de la largeur de la table. Notez l'usage du Variant et la manière de mettre à jour l'objet TableColumnSeparators.

```
rem Code11-05.sxw   bibli : LigneColonne Module3
Option Explicit

Sub ChangerLargeurDeColonne()
Dim monDocument As Object, maTable As Object
Dim lesSep As Variant, lesColonnes As Object
Dim largeur As Double
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Résultats")

largeur = MaTable.TableColumnRelativeSum
lesColonnes = maTable.Columns
lesSep = maTable.TableColumnSeparators
print "Nombre de colonnes : ", lesColonnes.Count
print "Avant : ", largeur, _
  lesSep(0).Position, lesSep(1).Position, lesSep(2).Position
lesSep(0).Position = largeur * 0.1
lesSep(2).Position = largeur * 0.675
maTable.TableColumnSeparators = lesSep
print "Après : ", largeur, _
  lesSep(0).Position, lesSep(1).Position, lesSep(2).Position
End Sub
```

Les colonnes sont numérotées à partir de zéro. Nous allons ajouter 3 colonnes adjacentes, dont la première aura le rang 1, en utilisant la méthode insertByIndex de l'objet Columns.

```
rem Code11-05.sxw   bibli : LigneColonne Module4
Option Explicit
```

```
Sub InsererColonnes()
Dim monDocument As Object, maTable As Object
Dim lesSep As Variant, lesColonnes As Object
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Résultats")
lesColonnes = maTable.Columns
lesColonnes.insertByIndex(1,3) ' ajouter 3 colonnes au rang 1
print "Maintenant on va supprimer"
lesColonnes.removeByIndex(1,3) ' supprimer ces trois colonnes
End Sub
```

L'ancienne colonne de rang 1 a été découpée en 4 colonnes, en gardant une largeur totale identique ; l'ancienne colonne de rang 1 est maintenant la colonne de rang 4.

Nous avons ensuite utilisé une méthode symétrique, `removeByIndex`, dont le premier argument est le rang de la première colonne à supprimer et le deuxième argument est le nombre de colonnes à supprimer.

Se déplacer dans un tableau

Les cellules d'un tableau Writer sont repérées par une adresse alphanumérique du genre « B7 », comme dans une feuille de classeur Calc. Ce repérage est valable pour des tableaux constitués d'un quadrillage régulier. On obtient ainsi l'objet cellule de l'emplacement « B7 » :

```
Dim maCellule As Object
maCellule = maTable.getCellByName("B7")
```

Le curseur de cellule

Il existe une notion de curseur de cellule, comme pour Calc. On le crée et on le positionne dans la même instruction :

```
Dim cursCell As Object
cursCell = maTable.createCursorByCellName("B7")
```

Le curseur de cellule permet d'effectuer quelques opérations sur l'ensemble de la cellule : couleur de fond, format de caractère...

Il existe plusieurs méthodes permettant de déplacer le curseur dans le tableau, voir tableau 11-26.

Tableau 11–26 Déplacement du curseur de cellule de tableau

Méthode	Signification
`goRight(n,SEL)`	Déplacer de n cellules à droite, avec saut de ligne éventuel.
`goLeft(n,SEL)`	Déplacer de n cellules à gauche, avec saut de ligne éventuel.

Tableau 11–26 Déplacement du curseur de cellule de tableau

Méthode	Signification
goUp(n,SEL)	Déplacer de n cellules vers le haut.
goDown(n,SEL)	Déplacer de n cellules vers le bas.
gotoStart(SEL)	Déplacer à la première cellule, en haut à gauche.
gotoEnd(SEL)	Déplacer à la dernière cellule, en bas à droite.
gotoCellByName(nom, SEL)	Aller à la cellule dont l'adresse alphanumérique est donnée en premier argument.

L'argument SEL de ces méthodes signifie :

- False : déplacement simple ;
- True : déplacer le curseur en sélectionnant les cellules.

La propriété RangeName du curseur de cellule fournit son adresse dans le tableau, sous forme d'une chaîne de caractères. Elle nous servira à obtenir la cellule pointée par le curseur. Cet exemple écrit un tableau 5 × 5 écrit en diagonale.

```
rem Code11-05.sxw   bibli : EcrireCellules Module1
Option Explicit

Sub EcrireEnDiagonale1()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, maTable As Object
Dim maCellule As Object, cursCell As Object
Dim x As Long
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
maTable = monDocument.createInstance(_
                      "com.sun.star.text.TextTable")
maTable.initialize(5,5) ' nombre de : lignes, colonnes
monTexte.insertTextContent(monCurseur, maTable, false)
cursCell = maTable.createCursorByCellName("E1")
for x = 1 to 5
  maCellule = maTable.getCellByName(cursCell.RangeName)
  maCellule.String = x      ' remplacer le contenu de la cellule
  cursCell.goDown(1, false) ' descendre d'une ligne
  cursCell.goLeft(1, false) ' reculer d'une cellule à gauche
next
End Sub
```

Zone de cellules

Nous pouvons sélectionner plusieurs cellules d'un tableau en déplaçant le curseur de cellule avec l'argument `true`. Exemple :

```
cursCell = maTable.createCursorByCellName("B3")
cursCell.goRight(1,true)
cursCell.goDown(1,true)
print cursCell.RangeName
```

Si vous exécutez la séquence, elle vous affichera « C4:B3 ». La zone sélectionnée est définie par les coordonnées des cellules en diagonale (B3:C4 serait équivalent).

Le curseur de cellule pointe maintenant sur une zone de cellules. Avec un tel curseur, on peut effectuer des opérations communes à toutes les cellules de la zone, par exemple des colorations.

Notez que `createCursorByCellName` ne sait pointer que vers une seule cellule, pas une zone. On utilise une autre méthode pour obtenir une zone de cellules :

```
Dim zoneCellule As Object
zoneCellule = maTable.getCellRangeByName("B2:C4")
zoneCellule.BackColor = RGB(255,204,255) ' couleur de fond
```

Se déplacer sans curseur de cellule

L'utilisation d'un curseur de cellule est assez pénible quand on souhaite accéder à des cellules dont on connaît les coordonnées en ligne et colonne. L'objet table nous fournit la méthode `getCellByPosition(x,y)`, qui nous renvoie la cellule désignée par les coordonnées.

L'exemple d'écriture en diagonale en est simplifié :

```
rem Code11-05.sxw   bibli : EcrireCellules Module2
Option Explicit

Sub EcrireEnDiagonale2()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, maTable As Object
Dim maCellule As Object, cursCell As Object
Dim x As Long
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
maTable = monDocument.createInstance(_
                        "com.sun.star.text.TextTable")
maTable.initialize(5,5) ' nombre de : lignes, colonnes
monTexte.insertTextContent(monCurseur, maTable, false)
```

```
for x = 0 to 4
  maCellule = maTable.getCellByPosition(4-x, x)
  maCellule.String = x ' remplacer le contenu de la cellule
next
End Sub
```

Zone de cellules

De même, la méthode `getCellRangeByPosition` nous renvoie une zone de cellules. Elle utilise quatre arguments pour préciser les coordonnées de la zone, respectivement :

1. le rang de la première colonne à gauche,
2. le rang de la première ligne en haut,
3. le rang de la dernière colonne à droite,
4. le rang de la dernière ligne en bas.

Exemple :

```
zoneCellule = maTable.getCellRangeByPosition(1,1,2,3)
```

Zone sélectionnée par l'utilisateur

Dans un tableau, l'utilisateur peut effectuer diverses sélections :

1. pointer un endroit dans le texte d'une cellule,
2. sélectionner une ou plusieurs zone(s) dans le texte d'une cellule,
3. sélectionner une ou plusieurs zone(s) dans les textes de plusieurs cellules,
4. sélectionner plusieurs cellules contiguës.

Nous disposons de deux objets pour distinguer ces cas :

```
laSel = monDocument.currentSelection
CurseurVisible = monDocument.currentcontroller.ViewCursor
```

Le curseur visible est la position du curseur clignotant. Il nous indique la table utilisée avec sa propriété `TextTable` et la cellule dans la table avec sa propriété `Cell`.

La sélection courante prend des apparences différentes suivant les cas de sélection indiqués plus haut.

1. L'objet `laSel` supporte le service `com.sun.star.text.TextRanges` (notez le « s » final). Sa propriété `Count` vaut 1. L'objet `laSel` donne accès à un seul élément : `laSel(0)`.
   Cet élément supporte le service `com.sun.star.text.TextRange`. Sa propriété `String` est une chaîne vide, sa propriété `Cell` nous donne la cellule pointée, sa propriété `TextTable` nous donne le tableau.

2 Ici, la propriété `Count` vaut n + 1 s'il y a n zones sélectionnées. Tous les éléments, auxquels on accède par indexation, supportent le service `com.sun.star.text.TextRange`. L'un d'eux est celui indiqué au point 1, les autres donnent dans leur propriété `String` une des zones sélectionnées.
3 Identique au cas 2, mais la propriété `Cell` des éléments indiquent des cellules différentes.
4 L'objet `laSel` est totalement différent ; il supporte le service `com.sun.star.text.TextTableCursor`. Sa propriété `RangeName` indique les coordonnées du rectangle de cellules sélectionnées, exemple : `"C3:B2"`.

Ainsi, gérer les sélections d'utilisateur dans un tableau n'est pas particulièrement simple. Et n'oubliez pas qu'il est possible de sélectionner des zones dans des cellules de tableau *et* ailleurs dans le document, par exemple dans le texte principal ou dans un autre tableau !

Les cellules

Une cellule de tableau Writer comporte de nombreuses propriétés. Le tableau 11-27 en liste les principales, d'autres seront décrites plus loin.

Tableau 11–27 Propriétés de cellule de tableau

Propriété	Type	Signification
`BackColor`	`Long`	Couleur du fond.
`BackTransparent`	`Boolean`	`True` rend le fond transparent (interface utilisateur : sans remplissage).
`TopBorder`	`Object`	Structure de la ligne de bordure du haut.
`TopBorderDistance`	`Long`	Espacement par rapport à la bordure du haut.
`BottomBorder`	`Object`	Structure de la ligne de bordure du bas.
`BottomBorderDistance`	`Long`	Espacement par rapport à la bordure du bas.
`LeftBorder`	`Object`	Structure de la ligne de bordure de gauche.
`LeftBorderDistance`	`Long`	Espacement par rapport à la bordure de gauche.
`RightBorder`	`Object`	Structure de la ligne de bordure de droite.
`RightBorderDistance`	`Long`	Espacement par rapport à la bordure de droite.
`CellName`	`String`	Coordonnées alphanumériques de la cellule.
`IsProtected`	`Boolean`	`True` si la cellule est protégée.

Bordures de cellule

Les propriétés de bordure de lignes (`TopBorder`, etc.) sont des structures déjà décrites, voir tableau 11-19.

Il faut définir une variable ayant cette structure, la remplir, puis l'affecter à la propriété de bordure :

```
Dim unBord As New com.sun.star.table.BorderLine

With unBord ' pour vérifier, utilisez le zoom à 200% !
  .Color = RGB(200,0,0)
  .OuterLineWidth = 30
  maCellule.LeftBorder = unBord ' ligne simple, rouge
  .OuterLineWidth = 100
  maCellule.RightBorder = unBord ' ligne simple, rouge
  .InnerLineWidth = 60
  .LineDistance = 30
  .Color = RGB(0,120,0)
  maCellule.TopBorder   = unBord ' ligne double, verte
  .Color = RGB(0,0,120)
  maCellule.BottomBorder = unBord ' ligne double, bleue
End With
```

Écrire un texte dans la cellule

Ayant obtenu accès à une cellule par les moyens décrits plus haut, on peut obtenir un curseur d'écriture *à l'intérieur de la cellule*. Les possibilités d'écriture sont alors exactement les mêmes que pour écrire du texte ordinaire. Cet exemple ajoute un texte en rouge et en gras à la fin du premier mot de la cellule A2 d'une table existante.

```
rem Code11-05.sxw   bibli : EcrireCellules Module3
Option Explicit

Sub InsererTexteDansCellule()
Dim monDocument As Object, maTable As Object
Dim maCellule As Object
Dim monCurseur As Object
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Résultats")
maCellule = maTable.getCellByName("A2")
monCurseur = maCellule.createTextCursor
monCurseur.gotoEndOfWord(false)
monCurseur.CharWeight = com.sun.star.awt.FontWeight.BOLD
monCurseur.CharColor = RGB(250,0,0) 'Rouge
maCellule.insertString(monCurseur, " 2004", false)
End Sub
```

La variable `maCellule` joue le même rôle que la variable `monTexte` dans les exemples d'écriture dans le texte principal. Tout le reste est identique, caractères spéciaux, style, formatage.

Formules et valeurs numériques

Les cellules d'un tableau Writer peuvent comporter des nombres ou des formules faisant référence au contenu de diverses cellules. Ne confondez pas avec un contenu texte qui représenterait un nombre ou une formule.

La propriété `Value` d'un objet cellule sert à affecter une valeur dans la cellule. Cette propriété est du type `Double`. Les nombres avec décimales sont écrits en Basic avec un point décimal, mais le tableau les affiche en tenant compte des spécificités nationales (la virgule en français).

On remplit une cellule avec une formule en utilisant la propriété `Formula`, de type `String`. L'aide en ligne de Writer explique comment obtenir des formules.

Cet exemple calcule la somme de plusieurs cellules dans le tableau « Finances ». Il montre aussi que la propriété `String` de la cellule renvoie alors la chaîne de caractères correspondant à la valeur numérique de la cellule.

```
rem Code11-05.sxw   bibli : EcrireCellules Module4
Option Explicit

Sub ValeursEtFormules()
Dim monDocument As Object, maTable As Object
Dim maCellule As Object
monDocument = ThisComponent
maTable = monDocument.TextTables.getByName("Finances")
maCellule = maTable.getCellByName("B2")
maCellule.Value = 2200.55
maCellule = maTable.getCellByName("B3")
maCellule.Value = 15260
maCellule = maTable.getCellByName("B4")
maCellule.Value = 15.75 + 22.3 + 2.35*4
print "Cellule B4 : " & maCellule.String
maCellule = maTable.getCellByName("B5")
maCellule.Formula = "sum(<B2:B4>)"
print "Résultat de la cellule B5 : " & maCellule.String
End Sub
```

L'expression affectée à la cellule B4 est en réalité calculée par Basic et la cellule ne reçoit que le nombre résultant.

Trier un tableau

Un objet tableau de Writer fournit une méthode de tri (en anglais : *Sort*). Cependant, cette méthode est peu utilisable directement, car elle trie tout le tableau alors qu'en général un tableau comporte des en-têtes de colonnes ou de lignes qui doivent rester inchangés dans le tri. En pratique, nous définirons un objet zone de cellules couvrant les cellules à trier dans le tableau. Cette zone de cellules fournit elle aussi la méthode de tri `Sort`.

> **ATTENTION Compatibilité**
>
> La version 1.1 d'OpenOffice.org a modifié le mécanisme de tri. Nous ne décrirons que ce dernier, qui n'est pas compatible avec la version précédente 1.0.x.

Nous allons effectuer un tri de la troisième colonne d'un tableau nommé `Triage`. Voici l'ensemble du codage, que nous expliquerons progressivement.

```
rem Code11-05.sxw   bibli : Trier Module1
Option Explicit

Sub TrierTableParCouleur()
Dim MonDocument As Object
Dim MaTable As Object, zoneDeTri As Object
Dim ConfigTri(0) As New com.sun.star.table.TableSortField
Dim DescrTri As Variant
MonDocument = ThisComponent
MaTable = MonDocument.TextTables.getByName("Triage")

With ConfigTri(0)
  .Field = 3 ' colonne C = "Couleurs"
  .IsAscending = true
  .IsCaseSensitive = MsgBox("Différencier la Casse ?", 4) = 6
  .FieldType = com.sun.star.table.TableSortFieldType.ALPHANUMERIC
End With
zoneDeTri = MaTable.getCellRangeByName("A2:D14")
DescrTri = zoneDeTri.createSortDescriptor
setPropVal(DescrTri, "SortFields", ConfigTri())
setPropVal(DescrTri, "IsSortColumns", false)
setPropVal(DescrTri, "IsSortInTable", true)
zoneDeTri.Sort(DescrTri())
End Sub

Sub setPropVal(descr As Variant, nomProp As String, valProp As Variant)
Const Titre = "Tableau de propriétés"
Dim x As Long
for x = 0 to UBound(descr)
  if descr(x).Name = nomProp then
    descr(x).Value = valProp
    Exit Sub
  end if
next
MsgBox("Propriété inconnue : " & nomProp, 16, Titre)
End Sub
```

La méthode de tri a besoin d'un descripteur de tri (notre variable `DescrTri`) qui précise les conditions globales du tri à effectuer. Parmi celles-ci, nous trouvons l'élément `SortFields` qui contient un tableau (`Array`) de descripteurs, un par colonne à trier (notre variable `ConfigTri`). Nous allons maintenant décrire chaque descripteur.

Le descripteur de tri par colonne est une structure composée des éléments détaillés dans le tableau 11-28.

Tableau 11–28 Descripteur de tri par colonne

Élément	Type	Signification
`Field`	`Long`	Rang de la colonne dans la zone de cellules ; valeur 1 pour la première colonne.
`IsAscending`	`Boolean`	`True` pour un tri par ordre croissant.
`IsCaseSensitive`	`Boolean`	`True` pour tenir compte de la casse des caractères.
`FieldType`	`Integer`	Type du champ. Constante nommée, décrite plus loin.
`CollatorLocale`	`Object`	Langue et variante nationale de langue.
`CollatorAlgorithm`	`String`	Méthode de tri.

L'élément `FieldType` a trois valeurs possibles, dont deux sont évidentes :

```
com.sun.star.table.TableSortFieldType.ALPHANUMERIC
com.sun.star.table.TableSortFieldType.NUMERIC
com.sun.star.table.TableSortFieldType.AUTOMATIC
```

Avec `AUTOMATIC`, OpenOffice.org décide lui-même si le type est numérique ou alphanumérique, alors autant ne pas l'utiliser.

L'élément `CollatorLocale` est lui-même une structure. Pour les cas courants (français, anglais, espagnol, allemand...), il n'est pas nécessaire de le remplir, ce qui nous évitera de le décrire. De même, `CollatorAlgorithm` peut être omis car il n'y a actuellement pas de choix possible (la seule valeur possible est : `alphanumeric`).

Une fois rempli le descripteur de tri par colonne, nous définissons la zone de tri dans le tableau, en excluant la première ligne qui contient les en-têtes de colonnes.

Le descripteur global de tri est obtenu avec la méthode `createSortDescriptor` de l'objet `zoneDeTri`. Ce descripteur est un tableau (`Array`) de structures `PropertyValue`. Chacune de celles-ci comporte un nom et une valeur. Le tableau 11-29 liste ces propriétés. Les noms sont initialisés par `createSortDescriptor`, mais l'ordre des propriétés dans le tableau de structures dépend de l'implémentation. La routine utilitaire `setPropVal` recherche la propriété d'un nom donné et lui affecte la valeur qui est en argument ; cette routine est décrite à l'annexe B.

L'élément `IsSortColumns`, au nom particulièrement mal choisi, indique si on trie par colonne (notre cas, le cas courant), ou si on trie des lignes (dans ce cas, le tableau aura des en-tête de lignes et tout se passe comme si on avait tourné le tableau de 90 degrés). Nous continuons sur notre exemple de tri de colonnes, sachant qu'il suffit de remplacer le terme « ligne » par « colonne » dans le deuxième cas.

Tableau 11–29 Descripteur global de tri

Élément	Type	Signification
`IsSortColumns`	`Boolean`	`False` pour trier par colonne (intervertir les lignes). `True` pour trier par lignes (intervertir les colonnes).
`SortFields`	`Array(Object)`	Tableau des descripteurs de tri par colonne.
`IsSortInTable`	`Boolean`	`True` pour trier un tableau.
`Delimiter`	`Integer`	Non utilisé pour le tri d'un tableau.
`MaxSortFieldsCount`	`Long`	En lecture seule, nombre maximal de colonnes à trier, valeur actuelle = 3.

L'élément `IsSortInTable` vaut ici toujours `True`. La valeur `False` et l'élément `Delimiter` sont utilisés pour des tris de paragraphes de texte, que nous ne développerons pas (nous n'avons pu le faire fonctionner).

L'élément `MaxSortFieldsCount` est pour nous une constante. Dans la version actuelle d'OpenOffice.org, on peut trier sur un maximum de 3 colonnes, comme on le verra dans l'exemple suivant.

Une fois le descripteur global de tri initialisé, la table est triée avec la méthode `Sort`. Si vous exécutez la macro sur le document fourni dans le zip téléchargeable, vous pourrez voir comment s'effectue le tri avec ou sans prise en compte de la casse. Pour revenir à l'ordre initial de la table, utilisez le bouton Annuler sur la fenêtre Writer, ou exécutez la macro `TrierTableParLignes` qui est dans le module 3 de la même bibliothèque.

Le tableau `Triage` est suffisamment rempli pour permettre de trier sur trois colonnes successivement. Nous utilisons maintenant un tableau de 3 descripteurs de colonnes, qui sont prises en compte dans le même ordre que ces descripteurs. Les deux premières colonnes contiennent des textes, la troisième des nombres.

```
rem Code11-05.sxw   bibli : Trier Module2
Option Explicit

Sub TrierTableSur3Colonnes()
Dim MonDocument As Object
Dim MaTable As Object, zoneDeTri As Object
Dim ConfigTri(2) As New com.sun.star.table.TableSortField
Dim DescrTri As Variant

MonDocument = ThisComponent
MaTable = MonDocument.TextTables.getByName("Triage")

ConfigTri(0).Field = 3 ' colonne C = "Couleur"
ConfigTri(0).IsAscending = true
ConfigTri(0).FieldType = com.sun.star.table.TableSortFieldType.ALPHANUMERIC
```

```
ConfigTri(1).Field = 2 ' colonne B = "Ville"
ConfigTri(1).IsAscending = true
ConfigTri(1).FieldType = com.sun.star.table.TableSortFieldType.ALPHANUMERIC

ConfigTri(2).Field = 4 ' colonne D = "Nombre"
ConfigTri(2).IsAscending = false
ConfigTri(2).FieldType = com.sun.star.table.TableSortFieldType.NUMERIC

zoneDeTri = MaTable.getCellRangeByName("A2:D14")
DescrTri = zoneDeTri.createSortDescriptor
setPropVal(DescrTri, "SortFields", ConfigTri())
setPropVal(DescrTri, "IsSortColumns", false)
setPropVal(DescrTri, "IsSortInTable", true)

zoneDeTri.Sort(DescrTri())
End Sub
```

Exécutez la macro et étudiez attentivement le résultat : il existe des lignes qui nécessitent les trois critères pour les classer.

> **BOGUE Prise en compte de la casse**
>
> D'après nos essais, la prise en compte de la casse ne fonctionne pas sur un tri à plusieurs colonnes.

Le tri de tableau est décrit dans le SDK à l'interface `XSortable` et aux services `SortDescriptor2` et `TableSortDescriptor2`.

Tableaux irréguliers

Jusqu'à maintenant, nous avons traité des tableaux constitués d'un quadrillage régulier. Partons par exemple d'un tableau de 5 lignes et 4 colonnes.

Pour des questions d'esthétique ou de titrage, il est parfois nécessaire de fusionner plusieurs cellules contiguës. Pour cela, on sélectionne la zone des cellules à fusionner, puis on utilise la méthode `mergeRange` :

```
cursCell = maTable.createCursorByCellName("B3")
cursCell.goRight(1, true)
cursCell.goDown(1, true)
cursCell.mergeRange ' fusionner la zone B3:C4
```

Si les cellules sélectionnées contenaient du texte, les paragraphes seront accolés dans la cellule résultante. Le problème, c'est que les coordonnées de cellules sont maintenant assez spéciales, comme le montre la figure 11-2. En fait, OpenOffice.org considère maintenant qu'il s'agit d'un tableau de 4 lignes, dont on a scindé horizontalement la cellule A3 et la cellule C3. Pour désigner une « sous-cellule », on utilise la coordonnée alphanumé-

rique de la cellule d'origine, et on lui ajoute le rang de colonne et le rang de ligne de la sous-cellule dans la cellule d'origine. Et ceci en comptant à partir de 1 ! Dans le tableau en exemple, la sous-cellule A3.1.2 est sur la première colonne et la deuxième ligne dans le quadrillage de l'ancienne cellule A3.

Figure 11–2
Coordonnées des cellules après une fusion

A1	*B1*	*C1*	*D1*
A2	B2	C2	D2
A3.1.1	B3		C3.1.1
A3.1.2			C3.1.2
A4	B4	C4	D4

ASTUCE **Voir les coordonnées d'une cellule**

Dans un tableau Writer, le fait de cliquer dans une cellule affiche ses coordonnées en bas à droite dans la fenêtre Writer.

À partir d'un curseur de cellule pointant sur une cellule ou une zone de cellules, on peut scinder horizontalement ou verticalement chaque cellule initiale en plusieurs sous-cellules :

```
cursCell.splitRange(2,true) ' scinder horizontalement
```

Le premier argument est le nombre de sous-cellules *créées* : deux dans l'exemple, donc on se retrouve avec 3 sous-cellules par cellule scindée ! Le deuxième argument vaut :

- `False` pour scinder verticalement,
- `True` pour scinder horizontalement.

Les figures 11-3 et 11-4 montrent le résultat de scissions respectivement horizontale et verticale.

Figure 11–3
Coordonnées des cellules après scission horizontale

A1	*B1*	*C1*	*D1*
A2	B2	C2	D2
A3	B3.1.1	C3.1.1	
	B3.1.2	C3.1.2	
	B3.1.3	C3.1.3	
A4	B4.1.1	C4.1.1	D4
	B4.1.2	C4.1.2	
	B4.1.3	C4.1.3	
A5	B5	C5	D5

Figure 11–4
Coordonnées des cellules après scission verticale

A1	*B1*			*C1*			*D1*
A2	B2			C2			D2
A3	B3	C3	D3	E3	F3	G3	H3
A4	B4	C4	D4	E4	F4	G4	H4
A5	B5			C5			D5

Ici, la méthode de repérage n'est plus cohérente avec la précédente. En fait, bien qu'on puisse répéter des opérations de fusion et scission, l'adressage résultant devient très complexe à gérer. De plus, les propriétés et méthodes de lignes et colonnes ne s'appliquent plus.

Les cadres

Les cadres de texte ont bien des usages dans un document Writer, par exemple pour placer librement une zone pouvant contenir du texte, une image, etc. En anglais, le mot « cadre » se traduit par « frame ».

> **JARGON Frame**
> La documentation API emploie souvent le mot *frame* pour désigner aussi une fenêtre élémentaire.

Insérer un cadre

On utilise un curseur pour désigner l'endroit d'insertion du cadre. La méthode `createInstance` de l'objet document fournit l'objet cadre. Il est ainsi répertorié dans l'ensemble des cadres du document. L'insertion effective du cadre sera faite par la méthode `insertTextContent` de l'objet texte.

```
rem Code11-06.sxw   bibli : AjoutCadre Module1
Option Explicit

Sub InsererUnCadre()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, monCadre As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
' monCurseur est dans le troisième paragraphe
monCadre = monDocument.createInstance("com.sun.star.text.TextFrame")
monCadre.Width = 10400 ' 104 mm largeur
monCadre.Height = 2530 ' 25,3 mm de haut
monTexte.insertTextContent(monCurseur, monCadre, false)
End Sub
```

L'argument de `createInstance` doit être écrit en respectant la casse.

Le troisième argument de la méthode `insertTextContent` signifie :

- `False` = insérer dans le texte ;
- `True` = écraser la zone actuellement sélectionnée par le curseur.

L'exemple utilisait le minimum d'instructions. En pratique, il faut préciser :

- si la hauteur du cadre s'adapte à son contenu ou non ;
- par rapport à quoi le cadre se positionne (l'ancrage du cadre) ;
- sa position, absolue par rapport à l'ancre ou relative.

Ces notions seront précisées un peu plus loin.

Insérer plusieurs cadres

À chaque insertion d'un cadre, il est nécessaire d'obtenir un nouvel objet `TextFrame`, même si on insère plusieurs fois le même cadre. Il faut aussi réinitialiser à chaque fois les propriétés du cadre. Dans cet exemple, on insère deux cadres dans deux paragraphes du document Writer :

```
rem Code11-06.sxw   bibli : AjoutCadre Module2
Option Explicit

Sub InsererPlusieursCadres()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, monCadre As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
' monCurseur est dans le troisième paragraphe
monCadre = monDocument.createInstance(_
                        "com.sun.star.text.TextFrame")
monCadre.Width = 10400 ' 104 mm largeur
monCadre.Height = 2530 ' 25,3 mm de haut
monTexte.insertTextContent(monCurseur, monCadre, false)
monCurseur.gotoNextParagraph(False)
' monCurseur est dans le quatrième paragraphe
monCadre = monDocument.createInstance(_
                        "com.sun.star.text.TextFrame")
monCadre.Width = 1500 ' 15 mm largeur
monCadre.Height = 3000 ' 30 mm de haut
monTexte.insertTextContent(monCurseur, monCadre, false)
End Sub
```

Trouver un cadre

Rappelons que tout cadre possède un nom (par défaut : `Cadre1`, `Cadre2`, etc. par ordre de création). Avec le Navigateur, on peut donner un nom plus spécifique à chaque cadre.

Supposons que dans un document existant, nous ayons à modifier le contenu d'un cadre nommé « Encadré ». L'objet document fournit la collection de ses cadres avec la fonction `getTextFrames` (notez le « s » du pluriel) et on obtient un cadre d'un nom particulier avec la

fonction `getByName` de l'objet collection. En Basic, on utilise simplement `TextFrames` et on accède à chaque cadre soit par un index, qui est un numéro d'ordre, soit par `getByName`.

```
monCadre = monDocument.TextFrames.getByName("Encadré")
```

Nous pouvons maintenant modifier le cadre en utilisant la variable `monCadre`.

Le `getByName` déclenchera une exception s'il n'existe aucun cadre de ce nom. Vous pouvez vérifier qu'un tel nom de cadre existe :

```
if monDocument.TextFrames.hasByName("Encadré") then
  ' le cadre existe, le modifier
end if
```

Notez qu'il est facile de renommer un cadre :

```
monCadre = monDocument.TextTables.getByName("Encadré")
monCadre.Name = "Cadre_allongé"
```

Un nom de cadre ne doit pas comporter de caractère espace.

L'objet `TextTables` nous permet de connaître tous les cadres du document et donc de les modifier. Le nombre de cadres est fourni par la propriété `Count`, leurs noms sont obtenus avec la propriété `Name` de chaque objet cadre.

```
rem Code11-06.sxw   bibli : AjoutCadre Module3
Option Explicit

Sub ListerLesCadres()
Dim monDocument As Object
Dim lesCadres As Object, monCadre As Object
Dim x As Long
monDocument = ThisComponent
lesCadres = monDocument.TextFrames
for x = 0 to lesCadres.Count -1
  monCadre = lesCadres(x)
  print "Cadre : " & monCadre.Name
next
End Sub
```

Attention : l'ordre des cadres dans la liste de `TextFrames` ne correspond pas nécessairement à l'ordre dans le document.

Supprimer un cadre

La méthode `removeTextContent` de l'objet texte fonctionne sur un cadre, à partir de la version 1.1.4 d'OpenOffice.org.

```
monTexte.removeTextContent(monCadre)
```

Dimensionner un cadre

La hauteur du cadre peut s'adapter ou non au contenu du cadre, par exemple si on y inscrit du texte sur plusieurs lignes. Ceci est défini par la propriété `SizeType`, de type `Integer`, qui reçoit une constante nommée (tableau 11-30), par exemple :

```
com.sun.star.text.SizeType.FIX
```

Tableau 11–30 Constantes de hauteur de cadre

Constante	Signification
`VARIABLE`	La hauteur dépend du contenu.
`FIX`	La hauteur est fixe, indiquée par `Height`.
`MIN`	La hauteur est au minimum celle indiquée par `Height` (valeur par défaut).

Positionner le cadre

Un cadre est positionné :

1. soit de manière absolue, en précisant la distance en 1/100 de millimètre entre le coin haut-gauche de l'ancre et le coin haut-gauche du cadre,
2. soit de manière relative, c'est-à-dire en décrivant sa situation par rapport à une zone qui englobe le cadre. Cette zone englobante doit aussi être précisée.

Le positionnement absolu ou relatif peut être choisi indépendamment pour la position horizontale et pour la position verticale.

Les nombreuses combinaisons possibles d'ancrage / zone englobante / position relative donnent des résultats parfois identiques, ou visibles seulement avec des tailles de cadre et de paragraphe compatibles. Faites des essais.

Si vous insérez manuellement un cadre dans un texte, un clic droit sur le cadre donne accès aux propriétés d'ancrage et d'alignement. L'alignement correspond au positionnement relatif du cadre.

Les différents ancrages de cadre

Le choix du type d'ancrage se fait en affectant à la propriété `AnchorType` de type `Integer`, une constante nommée (tableau 11-31), par exemple :

```
com.sun.star.text.TextContentAnchorType.AT_PAGE
```

Tableau 11–31 Constantes d'ancrage de cadre

Constante	Signification
AT_PARAGRAPH	Ancrage par rapport au paragraphe pointé par le curseur (valeur par défaut).
AS_CHARACTER	Ancrage comme si le cadre était un caractère ; la hauteur de la ligne s'adapte à la taille du cadre.
AT_PAGE	Ancrage par rapport à la page. `AnchorPageNo` contient le numéro de page ; par défaut, c'est la page où se trouve le curseur d'écriture.
AT_FRAME	Ancrage dans un cadre. `AnchorFrame` contient le cadre qui servira d'ancrage.
AT_CHARACTER	Ancrage par rapport au caractère pointé par le curseur.

Positionnement horizontal

Le positionnement dépend des propriétés de cadre `HoriOrient`, `HoriOrientRelation` et `AnchorType` que nous avons déjà vue.

La propriété `HoriOrient`, de type `Integer`, utilise des constantes nommées (tableau 11-32), par exemple :

```
com.sun.star.text.HoriOrientation.NONE
```

Tableau 11–32 Constantes de HoriOrient

Constante	Signification
NONE	Positionnement absolu. La position en 1/100 de mm est alors donnée par la propriété `HoriOrientPosition`
RIGHT	Positionnement relatif, à droite dans la zone englobante.
LEFT	Positionnement relatif, à gauche dans la zone englobante.
CENTER	Positionnement relatif, au centre dans la zone englobante.

La propriété `HoriOrientRelation`, de type `Integer`, précise la zone englobante. Elle utilise des constantes nommées, exemple :

```
com.sun.star.text.RelOrientation.PAGE_LEFT
```

Le tableau 11-33 indique les valeurs admises. La colonne repère est utilisée pour les explications qui suivent.

Tableau 11–33 Constantes de HoriOrientRelation

Constante	Repère	Signification
FRAME	a	Le paragraphe entier, y compris ses marges.
PRINT_AREA	b	Le paragraphe entier, sans ses marges.
CHAR	c	Le caractère.
PAGE_LEFT	d	La marge de gauche de la page ; en principe le cadre doit être assez petit pour tenir à l'intérieur de la marge
PAGE_RIGHT	e	La marge de droite de la page ; même remarque.
FRAME_LEFT	f	La marge de gauche du paragraphe ; même remarque.
FRAME_RIGHT	g	La marge de droite du paragraphe ; même remarque.
PAGE_FRAME	h	La page entière, y compris ses marges.
PAGE_PRINT_AREA	i	La page entière sans ses marges.

Le positionnement absolu est le plus simple : il suffit de préciser la distance en 1/100 de millimètre entre le coin haut-gauche de l'ancre et le coin haut-gauche du cadre.

```
monCadre.HoriOrient = com.sun.star.text.HoriOrientation.NONE
monCadre.HoriOrientPosition = 2500 ' 25 mm
```

Pour un positionnement relatif, `HoriOrientRelation` n'utilise que certaines valeurs, selon l'ancrage :

- pour un ancrage `AT_PARAGRAPH`, les valeurs repérées par a, b (par défaut), d, e, f, g ;
- pour un ancrage `AT_FRAME`, les valeurs repérées par a, b (par défaut), d, e, f, g ; dans ce contexte, il s'agit du paragraphe de texte à l'intérieur du cadre englobant ;
- pour un ancrage `AT_PAGE`, les valeurs repérées par d, e, h, i (par défaut).

Exemple de positionnement relatif :

```
monCadre.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
monCadre.HoriOrient = com.sun.star.text.HoriOrientation.RIGHT
monCadre.HoriOrientRelation = com.sun.star.text.RelOrientation.PAGE_LEFT
```

Positionnement vertical

Le positionnement dépend des propriétés de cadre `VertOrient`, `VertOrientRelation` et `AnchorType` que nous avons déjà vue.

La propriété `VertOrient`, de type `Integer`, utilise des constantes nommées (tableau 11-34), par exemple :

```
com.sun.star.text.VertOrientation.NONE
```

Tableau 11–34 Constantes de VertOrient

Constante	Signification
NONE	Positionnement absolu. La position en 1/100 de mm est alors donnée par la propriété `VertOrientPosition`
TOP	Positionnement relatif, en haut dans la zone englobante.
BOTTOM	Positionnement relatif, en bas dans la zone englobante.
CENTER	Positionnement relatif, au centre dans la zone englobante.

La propriété `VertOrientRelation`, de type `Integer`, précise la zone englobante. Elle utilise elle aussi des constantes nommées, par exemple :

```
com.sun.star.text.RelOrientation.FRAME
```

Le tableau 11-35 indique les valeurs admises. La colonne repère est utilisée pour les explications qui suivent.

Tableau 11–35 Constantes de VertOrientRelation

Constante	Repère	Signification
FRAME	a	Le paragraphe entier, y compris ses marges.
PRINT_AREA	b	Le paragraphe entier, sans ses marges.
CHAR	c	Le caractère.
PAGE_LEFT	d	La marge de gauche de la page ; en principe le cadre doit être assez petit pour tenir à l'intérieur de la marge
PAGE_RIGHT	e	La marge de droite de la page ; même remarque.
FRAME_LEFT	f	La marge de gauche du paragraphe ; même remarque.
FRAME_RIGHT	g	La marge de droite du paragraphe ; même remarque.
PAGE_FRAME	h	La page entière, y compris ses marges.
PAGE_PRINT_AREA	i	La page entière sans ses marges.

Le positionnement absolu est le plus simple : il suffit de préciser la distance en 1/100 de millimètre entre le coin haut-gauche de l'ancre et le coin haut-gauche du cadre.

```
monCadre.VertOrient = com.sun.star.text.HoriOrientation.NONE
monCadre.VertOrientPosition = 5500 ' 55 mm
```

Pour un positionnement relatif, `VertOrientRelation` n'utilise que certaines valeurs, selon l'ancrage :

- pour un ancrage AT_PARAGRAPH, les valeurs repérées par a, b (par défaut), d, e, f, g ;

- pour un ancrage AT_FRAME, les valeurs repérées par a, b (par défaut), d, e, f, g ; dans ce contexte, il s'agit du paragraphe de texte à l'intérieur du cadre englobant ;
- pour un ancrage AT_PAGE, les valeurs repérées par d, e, h, i (par défaut).

Exemple de positionnement relatif :

```
monCadre.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
monCadre.VertOrient = com.sun.star.text.VertOrientation.TOP
monCadre.VertOrientRelation = com.sun.star.text.RelOrientation.PAGE_LEFT
```

Maintenant, voici un exemple complet dans lequel on crée un petit cadre, situé au centre de la marge gauche de la page et verticalement en haut d'un paragraphe de plusieurs lignes qui sert d'ancrage.

```
rem Code11-06.sxw   bibli : PositionCadre Module1
Option Explicit

Sub PositionnerCadre()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, monCadre As Object
Dim x As Long

monDocument = ThisComponent
monCadre = monDocument.createInstance(_
                      "com.sun.star.text.TextFrame")
With monCadre
  .Width = 1000 ' 10 mm largeur
  .Height = 800 ' 8 mm de haut
  .SizeType = com.sun.star.text.SizeType.FIX
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
  .VertOrient = com.sun.star.text.VertOrientation.TOP
  .VertOrientRelation = com.sun.star.text.RelOrientation.FRAME
  .HoriOrient = com.sun.star.text.HoriOrientation.CENTER
  .HoriOrientRelation = com.sun.star.text.RelOrientation.PAGE_LEFT
End With

monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
for x = 1 to 5 ' aller au sixième paragraphe
  monCurseur.gotoNextParagraph(False)
next
monTexte.insertTextContent(monCurseur, monCadre, false)
monCadre.Text.String = "X"
End Sub
```

Pour apprécier les positionnements, essayez pour VertOrient les valeurs CENTER et BOTTOM, et pour HoriOrient les valeurs LEFT et RIGHT. Remarquez aussi que pour VertOrientRelation, les trois valeurs FRAME donnent le même résultat.

Adaptation du texte

La propriété `Surround`, de type `Integer`, correspond à « Adaptation du texte » quand on fait un clic droit sur un cadre sélectionné. Cette propriété reçoit une constante nommée (tableau 11-36) dont la valeur par défaut est :

```
com.sun.star.text.WrapTextMode.NONE
```

Tableau 11–36 Constantes de la propriété de cadre Surround

Constante	Signification
NONE	Pas d'adaptation. Valeur par défaut.
THROUGHT	Continu (le texte continue sous le cadre).
PARALLEL	Renvoi dynamique à la page (texte réparti à gauche et à droite du cadre)
DYNAMIC	Décidé par Writer selon la situation.
LEFT	Le texte est à gauche du cadre
RIGHT	Le texte est à droite du cadre.

La propriété `SurroundAnchorOnly`, de type `Boolean`, correspond à l'adaptation « Premier paragraphe ».

Autres propriétés de cadre

Dans le tableau 11-37, certaines des propriétés de cadre ont déjà été décrites pour les objets tableaux.

Tableau 11–37 Propriétés diverses des cadres

Propriété	Type	Signification
BackColor	Long	Couleur du fond.
BackTransparent	Boolean	True rend le fond transparent (interface utilisateur : sans remplissage).
BackColorTransparency	Integer	Pourcentage de transparence, valeur entre 0 et 100.
Opaque	Boolean	True rend le fond opaque. False place le cadre en arrière-plan.
TopBorder	Object	Structure de la ligne de bordure du haut. Voir bordure de tableau.
TopBorderDistance	Long	Espacement par rapport à la bordure du haut.
BottomBorder	Object	Structure de la ligne de bordure du bas. Voir bordure de tableau.
BottomBorderDistance	Long	Espacement par rapport à la bordure du bas.
LeftBorder	Object	Structure de la ligne de bordure de gauche. Voir bordure de tableau.
LeftBorderDistance	Long	Espacement par rapport à la bordure de gauche.

Tableau 11–37 Propriétés diverses des cadres (suite)

Propriété	Type	Signification
RightBorder	Object	Structure de la ligne de bordure de droite. Voir bordure de tableau.
RightBorderDistance	Long	Espacement par rapport à la bordure de droite.
ShadowFormat	Object	Ombre portée. Voir ombre de tableau.
FrameStyleName	String	Nom du style de cadre.

L'exemple qui suit insère un cadre semi-transparent de couleur cyan clair, ce qui permet de voir le texte principal au travers.

```
rem Code11-06.sxw   bibli : PositionCadre Module2
Option Explicit

Sub EntourageDuCadre()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, monCadre As Object
monDocument = ThisComponent
monCadre = monDocument.createInstance("com.sun.star.text.TextFrame")
With monCadre
  .Width = 5500 ' 55 mm largeur
  .Height = 2000 ' 20 mm de haut
  .BackColor = RGB(100,250,250)
  .BackColorTransparency = 40 ' transparent à 40%
  .Surround = com.sun.star.text.WrapTextMode.THROUGHT
End With
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextParagraph(False)
' monCurseur est dans le troisième paragraphe
monTexte.insertTextContent(monCurseur, monCadre, false)
monCadre.String = "Texte par dessus"
End Sub
```

Écrire du texte dans un cadre

Toutes les possibilités d'écriture et de formatage de texte que nous avons décrites pour le texte principal sont aussi disponibles pour écrire dans un cadre. Il suffit d'obtenir l'objet texte associé au cadre et de lui faire créer un curseur d'écriture :

```
Dim monTexte1 As Object, monCurseur1 As Object
rem - - initialisation et insertion du cadre - -
monTexte1 = monCadre.Text
monCurseur1 = monTexte1.createTextCursor
monTexte1.insertString(monCurseur1, "Blabla", false)
```

Les sections

Une section de texte est un ensemble de paragraphes contigus du texte qu'on peut traiter globalement, par exemple le protéger contre les modifications ou le rendre invisible. Une section peut contenir d'autres sections, et ainsi de suite.

Créer une section, écrire dedans

Le texte d'une section n'est pas séparé du texte principal, ainsi que va le démontrer cette macro. Exécutez-la ligne par ligne, en regardant attentivement le résultat dans le document.

```
rem Code11-14.sxw   bibli : Sections1 Module1
Option Explicit

Sub Creer1Section()
Dim monDocument As Object, monTexte As Object
Dim curseurT As Object, curseurS As Object
Dim maSection As Object, prgr As Integer
prgr = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
monDocument = ThisComponent
monTexte = monDocument.Text
curseurT = monTexte.createTextCursor ' curseur du texte courant
curseurT.gotoEnd(false)
curseurT.gotoPreviousParagraph(False)
monTexte.insertString(curseurT, "AAAA", false)
maSection = monDocument.createInstance(_
                        "com.sun.star.text.TextSection")
monTexte.insertTextContent(curseurT, maSection, false)
curseurS = monTexte.createTextCursorByRange(maSection.Anchor)
monTexte.insertString(curseurT, "BBBB", false)
monTexte.insertString(curseurS, "CCCC", false)
curseurT.gotoNextParagraph(False) ' sauter dans la section !
monTexte.insertString(curseurT, "DDDD", false)
curseurT.gotoNextParagraph(False) ' sauter après la section !
monTexte.insertString(curseurT, "EEEE", false)
monTexte.insertControlCharacter(curseurS, prgr, false)
monTexte.insertString(curseurS, "FFFF", false)
curseurS.gotoNextParagraph(False) ' sauter après la section !
monTexte.insertString(curseurS, "GGGG", false)
End Sub
```

Commençons par écrire « AAAA » dans l'avant-dernier paragraphe du texte principal. Pour créer une section, il nous faut un objet `TextSection` obtenu par la méthode `createInstance` de l'objet document. Cette nouvelle section est insérée à l'endroit indiqué par le curseur principal, `curseurT`. Vous constaterez qu'elle s'insère *après* le paragraphe en cours. Le `curseurT` n'est pas affecté par cette insertion, puisque la séquence « BBBB » s'insère après « AAAA ».

Une section possède une propriété `Anchor`, qui est une zone de texte (`TextRange`). Nous créons un nouveau curseur, `curseurS`, positionné sur cette zone, qui est actuellement réduite à un point car elle ne contient pas de texte. En utilisant `curseurS`, nous insérons le texte « CCCC » dans la section.

Reprenons `curseurT` pour le déplacer au paragraphe suivant. L'insertion du texte « DDDD » avec le `curseurT` apparaît au début de la section créée ! Déplaçons encore `curseurT` au paragraphe suivant. L'insertion du texte « EEEE » se place après la section ! Ceci démontre que, du point de vue d'un curseur, les sections n'existent pas, seuls les paragraphes existent.

Avec le `curseurS`, nous insérons un nouveau paragraphe : celui-ci est bien inséré dans la section, comme le montre l'écriture de « FFFF ». Cependant, si nous déplaçons le `curseurS` au paragraphe suivant, il se retrouve hors de la section, au début du paragraphe de texte principal, comme le montre l'insertion de « GGGG ».

Cet exemple un peu déroutant vous montre les difficultés à travailler sur des sections.

Naviguer dans les sections

Comment arriver à écrire sans se tromper sur un texte comportant des sections ?

D'une part, il faut savoir retrouver les sections dans un document existant. Un bon moyen consiste à leur donner un nom. Dans la fenêtre Writer, le nom de la section dans laquelle se trouve le curseur visible est affiché en bas à droite. Par programmation, une section se nomme avec sa propriété `Name` :

```
maSection.Name = "Annexe 2"
```

PIÈGE Nom de section existant

Si vous donnez par erreur à une section le nom d'une section existante, son nom ne sera pas modifié. Elle gardera le nom par défaut, `Section1` par exemple.

Le document Writer tient à jour la collection de ses sections dans l'objet `TextSections` (attention au « s »). Un élément de cette collection est accessible par son nom, avec les fonctions suivantes de l'objet collection :

- `hasByName`(nom) renvoie `True` si une section existe avec ce nom,
- `getByName`(nom) renvoie l'objet section de ce nom, ou déclenche une erreur s'il n'existe pas.

D'autre part, il est nécessaire de se positionner dans la section. Pour se positionner au début ou à la fin de la section, on utilise la propriété `Start` ou `End` de l'objet `Anchor` fourni par la section :

```
' créer un curseur au début de la section
curseurS = monTexte.createTextCursorByRange(maSection.Anchor.Start)
' déplacer le curseur à la fin de la section
curseurS.gotoRange(maSection.Anchor.End)
```

Un positionnement à un endroit quelconque peut être mémorisé par un signet (*bookmark*). Voyez ailleurs dans ce chapitre comment utiliser les signets.

Nous avons créé dans un document exemple trois sections nommées. La macro suivante va écrire dans chacune.

```
rem Code11-15.sxw   bibli : Sections2 Module1
Option Explicit

Sub NaviguerDansSections()
Dim monDocument As Object, monTexte As Object
Dim cursA As Object, cursB As Object, cursC As Object
Dim sectA As Object, sectB As Object, sectC As Object
Dim lesSections As Object, prgr As Integer
prgr = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
monDocument = ThisComponent
monTexte = monDocument.Text
lesSections = monDocument.TextSections
' il y aura une erreur si une des sections n'existe pas
sectA = lesSections.getByName("Aline")
sectB = lesSections.getByName("Brigitte")
sectC = lesSections.getByName("Claudine")
cursA = monTexte.createTextCursorByRange(sectA.Anchor.Start)
cursB = monTexte.createTextCursorByRange(sectB.Anchor)
cursC = monTexte.createTextCursorByRange(sectC.Anchor.End)
monTexte.insertString(cursA, "AAAA", false) ' insère au début
monTexte.insertString(cursB, "BBBB", true)  ' écrase le texte
monTexte.insertString(cursC, "CCCC", false) ' ajoute à la fin
monTexte.insertControlCharacter(cursA, prgr, false)
monTexte.insertControlCharacter(cursC, prgr, false)
monTexte.insertString(cursC, "DDDD", false)
print "Effacer la section Brigitte !"
cursB.gotoRange(sectB.Anchor, false) ' toute la section
cursB.String = "" ' efface son contenu
sectB.dispose' la section disparaît, mais reste un paragraphe
cursB.goLeft(1, true)
cursB.String = "" ' supprime le paragraphe précédent
End Sub
```

L'insertion de « BBBB » écrase tout le texte de la section, car le curseur `cursB` a été initialisé à la zone `Anchor`, ce qui équivaut à sélectionner tout le texte de la section.

La suppression complète d'une section demande un peu de gymnastique. D'abord, son contenu est supprimé. Toute la zone est remplacée par une chaîne de caractères vide. En fait, il va rester une marque de paragraphe dans la section. L'étape suivante consiste à

supprimer l'objet section, en invoquant sa méthode `dispose`. Elle ne supprime pas le texte correspondant (ici réduit à un paragraphe), qui devient un texte ordinaire, hors-section. Finalement, nous réalisons l'équivalent de se positionner dans ce paragraphe vide puis taper la touche Retour arrière. Si nous nous contentions de supprimer le paragraphe vide, c'est le paragraphe suivant qui deviendrait hors-section.

Si cette macro ne vous a pas posé de problème, alors vous maîtrisez bien la gestion des curseurs...

Propriétés des sections

Les principales propriétés de section de texte sont listées dans le tableau 11-38. D'autres sont documentées dans l'API, voir `com.sun.star.text.TextSection`.

Tableau 11–38 Propriétés de section

Élément	Type	Signification
`Name`	`String`	Nom de la section.
`FileLink`	`Object`	Le contenu de la section est obtenu d'un autre document.
`LinkRegion`	`String`	Nom d'une section ou d'un signet qui précise la zone de texte à utiliser pour la section.
`Condition`	`String`	Formule conditionnelle.
`IsVisible`	`Boolean`	`False` rend la section invisible.
`IsProtected`	`Boolean`	`True` pour protéger la section des modifications par l'utilisateur.
`ProtectionKey`	`Array`	Mot de passe crypté, un octet par élément.
`ParentSection`	`Object`	La section parente de celle-ci, ou `Null`.
`ChildSections`	`Array`	Tableau des sections incluses dans celle-ci, y compris les sous-sections.

La propriété `Condition` est évaluée pour décider si la section doit être invisible (si la propriété `IsVisible` est `False`). Si la propriété `Condition` est une chaîne vide, elle est considérée comme `True`. Les expressions conditionnelles sont expliquées dans l'aide en ligne de Writer, voir : Condition ; Champ.

La protection effectuée par `IsProtected` peut être annulée par l'utilisateur (utiliser le navigateur, clic droit sur la section, Éditer). L'API ne permet pas d'appliquer un vrai mot de passe, mais vous pouvez faire ceci :

```
Dim monSecret(2) As Integer
monSecret(0) = 25 ' exemple de faux mot de passe crypté
monSecret(1) = 26
monSecret(2) = -32
sectB.ProtectionKey = monSecret()
sectB.IsProtected = true
```

Le tableau `ProtectionKey` contient normalement les octets du mot de passe crypté. Ici, l'utilisateur devra retrouver le mot de passe correspondant, que personne ne connaît ! Vous cependant, en affectant `False` à `IsProtected`, vous pouvez déprotéger une section sans même connaître son mot de passe ! Pour supprimer le mot de passe, il suffit d'affecter à `ProtectionKey` un tableau vide.

La propriété `FileLink` est une structure composée de deux éléments :

- `FileURL`, de type `String`, est l'adresse du document lié,
- `FilterName`, de type `String`, est le nom du filtre éventuel nécessaire pour charger le fichier.

La propriété `LinkRegion` précise la zone de texte concernée dans le document lié. Si `FileLink` est vide, la zone de texte est recherchée dans le document principal.

Les styles

Un document Writer comporte des styles de paragraphe, de caractère, de cadre, de page et de numérotation. Il est de loin préférable de définir vos styles avec le styliste dans un document qui vous servira de modèle. Malgré tout, vous pouvez avoir à faire quelques modifications d'un style existant ; c'est ce que nous allons essentiellement décrire.

Trouver les styles

Les styles sont regroupés en famille, par exemple la famille des styles de paragraphe. Dans chaque famille, on trouve une collection de styles.

La macro suivante liste tous les styles d'un document. Comme cette liste est assez longue, nous allons l'écrire dans le document lui-même, ce qui nous fera un bon exercice d'écriture par macros.

```
rem Code11-07.sxw   bibli : ModifStyles Module1
Option Explicit

Sub ListerStyles()
Dim monDocument As Object
Dim monTexte As Object, monCurseur As Object
Dim lesFamilles As Object, uneFamille As Object
Dim styleX As Object, liste As String, nomFam As String
Dim f As Long, x As Long
Dim sautPage As Integer, sautLigne As String
sautPage = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
sautLigne = chr(10)
monDocument = ThisComponent
monTexte = monDocument.Text
```

```
monCurseur = monTexte.createTextCursor
monCurseur.gotoEnd(False) ' ajouter en fin de document
lesFamilles = monDocument.StyleFamilies
for f = 0 to lesFamilles.Count -1 ' chaque famille
  nomFam = lesFamilles.ElementNames(f)
  uneFamille = lesFamilles.getByName(nomFam)
  liste = "*** Famille " & nomFam
  for x = 0 to uneFamille.Count -1 ' chaque style
    styleX = uneFamille(x)
    liste = liste & sautLigne & chr(9) & styleX.Name _
                              & " = " & styleX.DisplayName
  next x
  monTexte.insertString(monCurseur, liste, false)
  monTexte.insertControlCharacter(monCurseur,sautPage,false)
next f
End Sub
```

La propriété `StyleFamilies` de l'objet document donne accès à toutes les familles de style qui existent dans le document. Le nombre de familles est la propriété `Count` de l'objet `lesFamilles` ; la liste des noms de familles est disponible dans le tableau `ElementNames` de ce même objet.

Bien qu'en Basic nous puissions obtenir successivement chaque famille par une simple indexation de l'objet `lesFamilles`, cet ordre n'est pas forcément le même que dans `ElementNames`. C'est pourquoi nous obtenons chaque famille en utilisant la fonction `getByName` de l'objet `lesFamilles`.

Le nombre de styles dans une famille est indiqué par la propriété `Count` de celle-ci. Nous obtenons chaque style par une simple indexation. Le nom du style est obtenu avec sa propriété `Name`. Les styles prédéfinis en standard ont un nom interne anglais. La macro indique aussi le nom localisé grâce à la propriété `DisplayName`.

Récupérer ou supprimer un style existant

Si vous connaissez le nom du style à modifier, et évidemment le nom de la famille, il sera facile de récupérer le style car les objets disposent d'un accès par le nom. Les familles de style ont pour nom :

```
ParagraphStyles    CharacterStyles     FrameStyles
PageStyles         NumberingStyles
```

Voici comment on récupère le style de page « Standard » :

```
Dim nomStyleMaPage As String, StyleMaPage As Object
Dim lesStylesPage As Object
lesStylesPage = monDocument.StyleFamilies.getByName("PageStyles")
StyleMaPage = lesStylesPage.getByName("Standard")
```

Notez que `getByName` fonctionne pour un nom de style anglais ou localisé.

Pour tester si un style existe dans une famille, on utilise la fonction `HasByName` :

```
if lesStylesPage.hasByName("HTML") then
  ' faire un traitement
end if
```

Pour supprimer un style personnel existant, on utilise la méthode `removeByName` :

```
lesStylesPage.removeByName("monStyle")
```

À défaut de créer votre document à partir d'un modèle comportant les styles souhaités (c'est la meilleure solution), vous pouvez copier dans votre document des styles existants dans un autre document. La méthode `loadStylesFromURL` de l'objet `StyleFamilies` copie par défaut l'ensemble des styles et écrase les styles existants du même nom. Cet exemple ne charge que les styles de page et ne modifie pas les styles existants.

```
rem Code11-07.sxw   bibli : ModifStyles Module2
Option Explicit

Sub ChargerStyles()
Dim monDocument As Object
Dim refDoc As String
Dim options(1) As New com.sun.star.beans.PropertyValue
options(0).Name = "LoadPageStyles"
options(0).Value = True ' ne charger que les styles de page
options(1).Name = "OverwriteStyles"
options(1).Value = False ' ne pas écraser les styles existants
refDoc = convertToURL("C:\Mes modeles\Doc2colonnes.stw")
monDocument = ThisComponent
monDocument.StyleFamilies.loadStylesFromURL(refDoc, options())
End Sub
```

On peut définir plusieurs options de chargement, comme l'indique le tableau 11-39.

Tableau 11–39 Propriétés optionnelles de chargement de style

Propriété	Type	Signification
`LoadTextStyles`	`Boolean`	`True` pour charger les styles de paragraphe et de caractère.
`LoadPageStyles`	`Boolean`	`True` pour charger les styles de page.
`LoadFrameStyles`	`Boolean`	`True` pour charger les styles de cadre.
`LoadNumberingStyles`	`Boolean`	`True` pour charger les styles de numérotation.
`OverwriteStyles`	`Boolean`	`True` pour charger tous les styles dans toutes les familles et écraser tous les styles existants.

Créer un nouveau style

La macro ci-dessous crée un nouveau style de paragraphe à partir d'un style existant, et le modifie avant de l'insérer dans la famille `ParagraphStyles`. Dans le Styliste, faire éventuellement basculer l'affichage d'une famille de style à l'autre pour faire apparaître le nouveau style.

```
rem Code11-07.sxw   bibli : ModifStyles Module3
Option Explicit

Sub HeriterStyle()
Dim monDocument As Object
Dim lesFamilles As Object, uneFamille As Object
Dim nouvStyle As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("ParagraphStyles")
nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.ParagraphStyle")
uneFamille.insertByName ("Signature colorée", nouvStyle)
nouvStyle.ParentStyle = "Signature" ' hériter d'un style
nouvStyle.CharColor = RGB(0,100,255) ' changer la couleur
End Sub
```

L'héritage d'un style avec la propriété `ParentStyle` ne fonctionne pas pour les styles de page et de numérotation car ils ne sont pas hiérarchisés. On obtient alors un style conforme au style `Standard`, qu'il faudra adapter à ses besoins.

Les propriétés de style

> **API Documentation des styles**
>
> Dans chaque famille de styles, nous ne listons dans les tableaux qui suivent que les propriétés les plus courantes. Si vos besoins sont plus spécifiques, il vous faudra étudier la documentation du SDK, à partir des liens de la page `com.sun.star.style`.

Un objet style possède une propriété `isInUse` qui renvoie `True` si ce style, ou un style dérivé, est effectivement utilisé dans le document.

Style de paragraphe

Le tableau 11-40 indique les principales propriétés de style de paragraphe ; de plus, un style de paragraphe possède toutes les propriétés de caractère, vues plus haut à la section formatage.

Tableau 11–40 Propriétés de style de paragraphe

Propriété	Type	Signification
ParaBackColor	Long	Couleur du fond.
ParaBackTransparent	Boolean	True rend le fond transparent (interface utilisateur : sans remplissage).
ParaAdjust	Integer	Alignement du texte. Constante nommée, voir tableau 11-41. Valeur par défaut : com.sun.star.style.ParagraphAdjust.LEFT
ParaFirstLineIndent	Long	Retrait de la première ligne, en 1/100 de mm.
ParaIsAutoFirstLineIndent	Boolean	True réalise un retrait automatique de la première ligne.
ParaStyleName	String	Nom du style.
FollowStyle	String	Nom du style appliqué au paragraphe inséré après celui-ci.
TopBorder	Object	Structure de la ligne de bordure du haut. Voir bordure de tableau.
TopBorderDistance	Long	Espacement par rapport à la bordure du haut.
BottomBorder	Object	Structure de la ligne de bordure du bas. Voir bordure de tableau.
BottomBorderDistance	Long	Espacement par rapport à la bordure du bas.
LeftBorder	Object	Structure de la ligne de bordure de gauche. Voir bordure de tableau.
LeftBorderDistance	Long	Espacement par rapport à la bordure de gauche.
RightBorder	Object	Structure de la ligne de bordure de droite. Voir bordure de tableau.
RightBorderDistance	Long	Espacement par rapport à la bordure de droite.
ParaShadowFormat	Object	Ombre portée. Voir ombre de tableau.
ParaTopMargin	Long	Écart avant le paragraphe, en 1/100 de mm.
ParaBottomMargin	Long	Écart après le paragraphe, en 1/100 de mm.
ParaLeftMargin	Long	Retrait avant le texte, en 1/100 de mm.
ParaRightMargin	Long	Retrait après le texte, en 1/100 de mm.
ParaTabStops	Object	Position des taquets de tabulation.
ParaKeepTogether	Boolean	True si ce paragraphe est solidaire avec le paragraphe suivant.
ParaSplit	Boolean	True si les lignes sont solidaires. False pour utiliser le traitement des veuves et orphelines.
ParaOrphans	Integer	Nombre de lignes orphelines.
ParaWidows	Integer	Nombre de lignes veuves.
ParaIsNumberingRestart	Boolean	True si la numérotation redémarre.
NumberingStartValue	Integer	Valeur initiale en cas de redémarrage de numérotation.
BreakType	Integer	Saut de page ou de colonne, constante nommée identique à la description générale.

Tableau 11–41 Constantes d'alignement de paragraphe

Constante	Signification
LEFT	Texte cadré à gauche.
RIGHT	Texte cadré à droite.
CENTER	Texte centré.
STRETCH	Texte justifié.
BLOCK	Texte justifié, sauf la dernière ligne qui dépend alors de ParaLastLineAdjust

Style de caractère

Un style de caractère possède toutes les propriétés de caractère, plus celles du tableau 11-42.

Tableau 11–42 Propriétés de style de caractère

Propriété	Type	Signification
CharDiffHeight	Double	Différence de taille, en points, du caractère par rapport au caractère parent.
CharPropHeight	Integer	Hauteur du caractère en pourcentage de celle du caractère parent. Exemple : 125 pour 125 %.

Style de page

Les propriétés de style de page comprennent, outre celles du tableau 11-43, des propriétés relatives à l'en-tête et au bas de page, qui sont décrites dans une section particulière.

Tableau 11–43 Propriétés de style de page

Propriété	Type	Signification
BackColor	Long	Couleur du fond.
BackTransparent	Boolean	True rend le fond transparent (interface utilisateur : sans remplissage).
FollowStyle	String	Nom du style appliqué à la page insérée après celle-ci.
TopBorder	Object	Structure de la ligne de bordure du haut. Voir bordure de tableau.
TopBorderDistance	Long	Espacement par rapport à la bordure du haut.
BottomBorder	Object	Structure de la ligne de bordure du bas. Voir bordure de tableau.
BottomBorderDistance	Long	Espacement par rapport à la bordure du bas.
LeftBorder	Object	Structure de la ligne de bordure de gauche. Voir bordure de tableau.
LeftBorderDistance	Long	Espacement par rapport à la bordure de gauche.

Tableau 11–43 Propriétés de style de page (suite)

Propriété	Type	Signification
RightBorder	Object	Structure de la ligne de bordure de droite. Voir bordure de tableau.
RightBorderDistance	Long	Espacement par rapport à la bordure de droite.
ShadowFormat	Object	Ombre portée. Voir ombre de tableau.
TopMargin	Long	Marge du haut, en 1/100 de mm.
BottomMargin	Long	Marge du bas, en 1/100 de mm.
LeftMargin	Long	Marge de gauche, en 1/100 de mm.
RightMargin	Long	Marge de droite, en 1/100 de mm.
IsLandscape	Boolean	True si la page est en orientation Paysage
NumberingType	Integer	Type de numérotation par défaut. Constante nommée, voir tableau 11-45. Valeur par défaut : com.sun.star.style.NumberingType.ARABIC
PageStyleLayout	Integer	Indique quelles pages sont concernées. Constante nommée, voir tableau 11-44. Valeur par défaut : com.sun.star.style.PageStyleLayout.ALL
PrinterPaperTray	String	Nom du bac de l'imprimante sélectionnée.
Width	Long	Hauteur de la page, en 1/100 de mm.
Height	Long	Largeur de la page, en 1/100 de mm.

Tableau 11–44 Constantes de disposition de page

Constante	Signification
ALL	Pages gauches et droites.
LEFT	Pages gauches seulement.
RIGHT	Pages droites seulement.
MIRRORED	Les pages gauches utilisent ce style, les pages droites prennent des valeurs "en miroir".

Tableau 11–45 Constantes de type de numérotation

Constante	Exemple de numérotation
ARABIC	1, 2, 3, 4
CHARS_UPPER_LETTER	A, B, C, D
CHARS_LOWER_LETTER	a, b, c, d
ROMAN_UPPER	I, II, III, IV, V
ROMAN_LOWER	i, ii, iii, iv, v

Tableau 11–45 Constantes de type de numérotation (suite)

Constante	Exemple de numérotation
NUMBER_NONE	Pas de numérotation.
CHARS_UPPER_LETTER_N	A, B, ..., Y, Z, AA, BB, CC, ... AAA, ...
CHARS_LOWER_LETTER_N	a, b, ..., y, z, aa, bb, cc, ... aaa, ...

Comment utiliser les styles de page

Pour insérer dans un document des pages d'orientation différente ou ayant d'autres caractéristiques particulières, il est recommandé d'utiliser des styles de page appropriés. Tous les styles de page sont basés sur le style `Standard`. Nous allons créer deux styles :

- un style à orientation Portrait au format A4, qui est simplement une copie du style `Standard` ;
- un style à orientation Paysage, lui aussi au format A4.

La macro va créer les deux styles successivement. Dans le Styliste, basculer éventuellement l'affichage d'une famille de style à l'autre pour faire apparaître les nouveaux styles.

```
rem Code11-07.sxw   bibli : OrientationPages Module1
Option Explicit

Sub CreerDeuxStylesPage()
Dim monDocument As Object
Dim lesFamilles As Object, uneFamille As Object
Dim nouvStyle As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("PageStyles")
nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.PageStyle")
' créer un nouveau style identique au style Standard
uneFamille.insertByName ("A4 Portrait", nouvStyle)

nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.PageStyle")
' créer un style Paysage
nouvStyle.IsLandscape = True
nouvStyle.Width = 29700 ' hauteur 29,7 cm
nouvStyle.Height = 21000 ' largeur 21,0 cm
' ( par défaut le style de suite est le style en cours )
uneFamille.insertByName ("A4 Paysage", nouvStyle)
End Sub
```

Pour chaque nouveau style, nous obtenons un objet style avec la fonction `CreateInstance` de l'objet document. Après adaptation éventuelle, ce style est inséré dans sa famille par la méthode `insertByName`.

Remarquez qu'avec le format Paysage nous avons aussi changé la hauteur et la largeur, alors qu'avec l'interface utilisateur un simple clic sur le bouton Paysage suffit à intervertir ces deux valeurs.

Nous disposons maintenant de deux nouveaux styles, « A4 Portrait » et « A4 Paysage ». Nous allons les utiliser dans la macro suivante, qui va ajouter plusieurs pages à un document existant, qui n'utilise au départ que des pages de style Standard.

```
rem Code11-07.sxw   bibli : OrientationPages Module2
Option Explicit

Sub AjouterPages()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
Dim PageAvant As Integer, paragr As Integer
paragr = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
PageAvant = com.sun.star.style.BreakType.PAGE_BEFORE
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
With monCurseur
  .gotoEnd(False)
  ' ajouter un paragraphe
  monTexte.insertControlCharacter(monCurseur, paragr, false)
  .PageDescName = "A4 Paysage" ' provoque un saut de page
  monTexte.insertString(monCurseur, "Style A4 Paysage", false)
  monTexte.insertControlCharacter(monCurseur, paragr, false)
  .breakType = PageAvant ' insère un saut de page AVANT
  monTexte.insertString(monCurseur, "Style A4 Paysage", false)
  monTexte.insertControlCharacter(monCurseur, paragr, false)
  .PageDescName = "Standard" ' provoque un saut de page
  monTexte.insertString(monCurseur, "Style Standard", false)
  monTexte.insertControlCharacter(monCurseur, paragr, false)
  .PageDescName = "A4 Portrait" ' provoque un saut de page
  monTexte.insertString(monCurseur, "Style A4 Portrait", false)
  monTexte.insertControlCharacter(monCurseur, paragr, false)
End With
End Sub
```

Nous allons en fin de document pour ajouter un paragraphe. En modifiant la propriété `PageDescName` du curseur, nous affectons le style de page « A4 Paysage » à partir de la position du curseur. Ceci provoque un saut de page.

Nous insérons un texte, puis un nouveau paragraphe dans cette page Paysage. La propriété `BreakType` insère un saut de page ordinaire, mais comme le style de page de suite est le même, nous restons en Paysage. Nous insérons encore un texte et un paragraphe.

Un nouveau changement de la propriété `PageDescName` nous fait revenir au style de page `Standard`, donc en orientation Portrait avec un nouveau saut de page. Nous insérons encore un texte et un paragraphe.

Enfin, nous changeons le style de page en « A4 Portrait » qui n'est qu'une copie du style Standard. Cela provoque néanmoins un nouveau saut de page.

> **ASTUCE Style de la page courante**
>
> Sur un document, vous pouvez facilement vérifier le style de la page où se trouve le curseur visible : il est affiché en bas de la fenêtre Writer, entre l'indicateur de numéro de page et celui du zoom. Un double-clic dans cette zone affiche le panneau de style de page

Style de cadre, style de numérotation

Les propriétés de style de cadre sont celles déjà décrites à la section cadre.

Les propriétés de style de numérotation sont constituées d'éléments « à tiroirs » qu'il serait trop long de décrire dans ce livre. Veuillez consulter le SDK sur les services suivants :

- `com.sun.star.text.NumberingRules`,
- `com.sun.star.text.NumberingLevel`.

Les en-têtes et pieds de page

Les en-têtes et pieds de page ne sont pas des éléments ordinaires du document. En effet, ils sont des éléments appartenant à un style de page, par défaut le style de page « `Standard` ». Pour faire apparaître ou pour modifier un en-tête ou pied de page, il faut retrouver le style de page de la page qui nous intéresse et le modifier. En conséquence, toutes les pages du même style subiront la même modification d'en-tête ou pied de page.

Nous allons insérer un en-tête dans la première page d'un document (et celles qui ont le même style).

```
rem Code11-07.sxw   bibli : Haut_Bas Module1
Option Explicit

Sub InsererUnEnTete()
Dim monDocument As Object
Dim monTexte As Object, monCurseur As Object
Dim   Texte2 As Object,   Curseur2 As Object
Dim nomStyleMaPage As String, StyleMaPage As Object
Dim lesStylesPage As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
' récupérer le nom du style de page en cours
nomStyleMaPage = monCurseur.PageStyleName
print "Cette page est du style : " & nomStyleMaPage
```

```
' récupérer la collection de styles de pages
lesStylesPage = monDocument.StyleFamilies.getByName("PageStyles")
' récupérer le style de la page en cours
StyleMaPage = lesStylesPage.getByName(nomStyleMaPage)
With StyleMaPage
  .HeaderIsOn = true ' insérer un en-tête
  .HeaderBodyDistance = 1000 ' 10 mm
  .HeaderHeight = 2500 ' 25mm
  Texte2 = .HeaderText ' zone de texte de l'en-tête
End With
Curseur2 = Texte2.createTextCursor ' curseur dans l'en-tête
' écrire un texte dans l'en-tête
Texte2.insertString(Curseur2, "Voici un en-tête", false)
End Sub
```

Nous voyons encore un exemple où l'usage d'un document modèle avec des styles bien définis peut alléger considérablement l'écriture de macros en évitant de créer ou modifier les en-têtes ou bas de page.

Tableau 11–46 Propriétés d'en-tête de page

Propriété	Type	Signification
HeaderIsOn	Boolean	True si un en-tête est activé.
HeaderIsShared	Boolean	True si les en-têtes gauche et droite sont identiques.
HeaderBackColor	Long	Couleur du fond.
HeaderBackTransparent	Boolean	True rend le fond transparent (interface utilisateur : sans remplissage).
HeaderLeftBorderDistance	Long	Distance entre marge et bord gauche de l'en-tête, en 1/100 de mm.
HeaderRightBorderDistance	Long	Distance entre marge et bord droit de l'en-tête, en 1/100 de mm.
HeaderTopBorderDistance	Long	Distance entre marge et bord du haut de l'en-tête, en 1/100 de mm.
HeaderBottomBorderDistance	Long	Distance entre marge et bord du bas de l'en-tête, en 1/100 de mm.
HeaderBodyDistance	Long	Distance entre en-tête et texte principal, en 1/100 de mm.
HeaderHeight	Long	Hauteur de l'en-tête, en 1/100 de mm.
HeaderShadowFormat	Object	Ombre portée. Voir ombre de tableau.
HeaderText	Object	Zone du texte, si les en-têtes gauche et droite sont identiques.
HeaderTextLeft	Object	Zone du texte de l'en-tête gauche.
HeaderTextRight	Object	Zone du texte de l'en-tête droite.
HeaderLeftMargin	Long	Marge gauche de l'en-tête.
HeaderRightMargin	Long	Marge droite de l'en-tête.
HeaderLeftBorder	Object	Structure de la ligne de bordure de gauche. Voir bordure de tableau.
HeaderRightBorder	Object	Structure de la ligne de bordure de droite. Voir bordure de tableau.
HeaderTopBorder	Object	Structure de la ligne de bordure du haut. Voir bordure de tableau.
HeaderBottomBorder	Object	Structure de la ligne de bordure du bas. Voir bordure de tableau.

L'objet style de page expose un très grand nombre de propriétés pour l'en-tête (`Header`) et pour le pied de page (`Footer`). Comme ces propriétés sont similaires, le tableau 11-46 ne liste que les principales propriétés de l'en-tête, sachant qu'il suffit de remplacer `Header` par `Footer` dans le nom de propriété pour obtenir celles du bas de page.

Les champs de texte

Les champs de texte, ou `TextField`, sont obtenus avec la fonction `createInstance` de l'objet document, puis insérés avec la méthode `insertTextContent` de l'objet texte après avoir rempli les propriétés. Voici un exemple qui insère le nom du fichier (sans son extension) dans le document.

```
rem Code11-13.sxw   bibli : ChampsTexte Module1
Option Explicit

Sub AjouterChampTexte()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
Dim monChamp As Object
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoEnd(false)
monChamp = monDocument.createInstance(_
                        "com.sun.star.text.TextField.FileName")
' attention bogue sur FileFormat quand IsFixed = true !!
monChamp.FileFormat = com.sun.star.text.FilenameDisplayFormat.NAME
monChamp.IsFixed = false ' mettre à jour le champ
monTexte.insertTextContent(monCurseur, monChamp, False)
End Sub
```

La propriété `FileFormat` reçoit une constante nommée, dont les valeurs possibles sont listées au tableau 11-47. L'ennui, c'est que lorsque la propriété `IsFixed` vaut `True`, une bogue mélange la signification des constantes ; dans ce cas, on doit utiliser une valeur numérique (la colonne de droite dans le tableau).

Tableau 11–47 Constantes de FileFormat

Constante pour IsFixed = false	Contenu du champ	Valeur pour IsFixed = true
FULL	Le nom complet avec le chemin.	1
PATH	Le chemin seulement.	2
NAME	Le nom seulement, sans l'extension.	3
NAME_AND_EXT	Le nom du fichier avec l'extension.	0

Les différents champs de texte sont décrits dans les branches de l'API :

- `com.sun.star.text.TextField`
- `com.sun.star.text.FieldMaster`

Une présentation exhaustive est donnée dans le guide du développeur (« Developer's Guide ») au chapitre 7.3.5. Chaque `TextField` a ses propriétés, pas toujours évidentes, voir dans la documentation API.

Certains champs `TextField` sont accessibles directement, d'autres sont regroupés par familles dans un `FieldMaster`. Plusieurs types d'informations et d'insertions sont possibles ; parmi eux, nous pouvons noter les champs utilisateurs, les notes ou les textes conditionnels.

Variables de champ utilisateur

Un champ utilisateur peut être considéré comme une variable du document, affichée ou non. Les champs couramment utilisés avec le menu Insertion > Champ sont en fait des objets `TextField` que l'on peut insérer dans le document.

L'API permet d'accéder aux mêmes possibilités que celles offertes par les différents onglets du menu Insertion>Champ>Autres.

Nous donnons à titre d'exemple quelques manipulations de ces champs.

Champs d'utilisateur

Ils sont situés dans l'onglet Variable. Le principe est de définir un champ et de l'ajouter à la liste en lui donnant un nom et une valeur. C'est un champ maître (ou `TextFieldMaster`). Le champ texte visible dans le document sera alors lié à ce champ maître. On pourra créer autant de `TextField` dépendant de ce `TextFieldMaster` que nécessaire. Si la valeur doit être mise à jour, seul le champ maître aura à être modifié.

Manipuler les champs d'utilisateur se fait donc en deux étapes :

- créer les champs maîtres,
- créer et insérer les champ dépendants.

Créer et modifier un champ maître TextFieldMaster

Il y a de nombreuses catégories de champs `TextFieldMaster`. Nous nous intéressons ici à la famille des champs utilisateur, `User` en anglais.

La macro suivante crée un champ utilisateur `MonChampMaitre` et lui affecte une valeur.

```
sub creerChamps()
  leDoc = thisComponent
  racineChampMaitre = "com.sun.star.text.FieldMaster.User"
  nomChamp = "MonChampMaitre"
```

```
  If not leDoc.TextFieldMasters.hasByName( _
              racineChampMaitre + "." + nomChamp) then
    leChampMaitre = leDoc.createInstance(racineChampMaitre)
    leChampMaitre.Name = nomChamp
  else
    leChampMaitre = leDoc.TextFieldMasters.getByName( _
                        racineChampMaitre + "." +  nomChamp)
  end if
  leChampMaitre.content = "la Valeur du champ"
end sub
```

Nous commençons par définir la variable racineChampMaitre qui contient la racine commune des noms des champs maîtres utilisateur :

```
com.sun.star.text.FieldMaster.User
```

La collection des champs maîtres `TextFieldMasters` est accessible depuis l'objet document `leDoc`. Comme toute collection, elle possède la méthode `hasByName` permettant de contrôler que le champ que nous désirons créer n'existe pas déjà. En effet, une exception, donc une erreur de la macro, est levée si l'on tente d'insérer une champ maître portant un nom déjà existant. Le nom d'un champ maître est constitué de sa racine, ici racineChampMaitre et de son nom, ici nomChamp, ce qui nous conduit à un champ maître nommé :

```
com.sun.star.text.FieldMaster.User.MonChampMaitre
```

C'est cette valeur qui doit être contrôlée. Si ce champ maître n'existe pas, nous créons, avec `createInstance`, un exemplaire de champ maître conteneur, donc un `MasterField` de la famille `User`, car notre racine est : `com.sun.star.text.FieldMaster.User`.

Nous obtenons donc le TextFieldMaster, un champ maître de type `User`. Il nous suffit de le nommer en utilisant sa propriété `Name` pour qu'il soit créé.

Si la méthode `hasByName` nous indique que le champ existe déjà, nous pouvons juste le récupérer à l'aide de la méthode `getByName` de la collection des `TextFieldMasters`, en utilisant le nom complet du champ.

Enfin, le contenu du champ maître leTextFieldMaster est simplement mis à jour en utilisant la propriété `Content`.

Insérer un champ utilisateur TextField

Une fois notre champ maître créé, nous pouvons l'insérer dans le document comme nous le ferions dans l'interface graphique. Nous choisissons ici de l'insérer au début du document.

```
Sub insertChamp()
  leDoc = thisComponent
  racineChampMaitre = "com.sun.star.text.FieldMaster.User"
```

```
  nomChamp = racineChampMaitre + ".MonChampMaitre "
  leChampMaitre = leDoc.TextFieldMasters.getByName(nomChamp)

  leChamp = leDoc.createInstance("com.sun.star.text.TextField.User")
  leChamp.attachTextFieldMaster(leChampMaitre)

  leCurseur = leDoc.Text.createTextCursor()
  leDoc.Text.insertTextContent(leCurseur, leChamp, false)

  leDoc.TextFields.refresh()
End Sub
```

Nous commençons à définir le nom du champ maître que nous voulons utiliser : `com.sun.star.text.FieldMaster.User.MonChampMaitre`. Nous utilisons ce nom pour appeler la méthode `getByName` de l'objet `TextFieldMasters` du document.

Nous créons ensuite, à l'aide de `createInstance`, un exemplaire de champ texte du même type que celui du champ maître que nous voulons utiliser ; dans notre cas : `User`. L'objet `TextField` à utiliser est donc `com.sun.star.text.TextField.User`.

À présent, nous disposons d'un objet `TextField` vierge, un genre de modèle. Il nous faut l'associer à un champ maître en utilisant sa méthode `attachTextFieldMaster`, l'argument de la méthode étant l'objet `FieldMaster` que nous avons récupéré auparavant.

Enfin, il nous suffit d'insérer ce champ dans le document en utilisant la méthode `insertTextContent` de l'objet `Text` du document comme nous l'avons vu précédemment.

Les champs du document doivent être ensuite rafraîchis en utilisant la méthode `TextFields.refresh` pour que les changements soient reflétés dans l'interface utilisateur.

Insérer un champ de base de données

L'API permet, comme nous le verrons plus loin dans ce livre, de manipuler des sources de données et d'accéder à des champs issus de requêtes ou de tables de bases de données.

Avec l'interface utilisateur, vous insérez les champs de ces sources de données en vue de publipostage par exemple. Sans rentrer dans le détail des sources de données, nous indiquons ici comment insérer un tel champ au moyen de l'API, le processus étant très voisin de celui d'un champ utilisateur.

```
sub insertChampBase()
leDoc = thisComponent

racineChampMaitre = "com.sun.star.text.FieldMaster.Database"
leChampMaitre = leDoc.createInstance(racineChampMaitre)

leChampMaitre.dataBaseName = "Bibliography"
leChampMaitre.dataTableName = "biblio"
leChampMaitre.dataColumnName = "Identifier"
```

```
leChamp = leDoc.createInstance("com.sun.star.text.TextField.Database")
leChamp.attachTextFieldMaster(leChampMaitre)

leCurseur = leDoc.Text.createTextCursor()
leDoc.Text.insertTextContent(leCurseur, leChamp, false)

leDoc.Textfields.refresh()
end sub
```

Un champ de base de données dépend d'un champ maître de type `Database`.

L'exemplaire de champ que nous obtenons par `createInstance` présente trois propriétés que nous devons renseigner :

- `dataBaseName` : le nom de la source de données ;
- `dataTableName` : le nom de la table concernée ;
- `dataColumnName` : le nom du champ dans la table.

Le champ maître est alors entièrement défini et porte le nom :

```
com.sun.star.text.FieldMaster.Database.Bibliography.biblio.Identifier
```

De même que précédemment, nous pouvons alors insérer, à l'aide de `createInstance`, un champ `TextField` de la même famille que le `TextFieldMaster` que nous avons défini, à savoir `com.sun.star.text.TextField.Database`, et lui attacher l'objet `TextFieldMaster` que nous avons créé pour notre champ de source de données, à l'aide de la méthode `attachTextFieldMaster`.

Le champ est ensuite inséré dans le document. Nous appelons enfin la méthode `refresh` de la collection `TextFields` pour mettre à jour le document.

Notes

Les notes sont de petits rectangles jaunes insérés au fil du texte, contenant des informations que l'utilisateur peut visualiser en double-cliquant dessus. Du point de vue de l'API, ce sont des `TextField` représentant un contenu de texte avec un auteur et une date comme propriétés supplémentaires :

```
rem Code11-13.sxw   bibli : ChampsTexte Module2
Option Explicit

Sub AjouterNote()
dim maNote As Object
dim i As Long, chaine As String
dim uneDate As New com.sun.star.util.Date
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object
```

```
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
for i=1 to 5
  maNote = monDocument.createInstance("com.sun.star.text.TextField.Annotation")
  maNote.Author = "un Auteur" ' auteur de la note
  ' contenu de la note, en deux lignes
  maNote.Content = "Le texte" & chr(13) & "de la note " & i

  uneDate.Day = 15     ' date de la note : jour
  uneDate.Month = 7    ' date de la note : mois
  uneDate.Year = 2004  ' date de la note : année
  maNote.Date = uneDate
  monTexte.insertTextContent(monCurseur, maNote, False)
  chaine = String(10, CStr(i)) ' ajouter du texte
  monTexte.insertString(monCurseur, chaine, False)
next i
End Sub
```

Une note étant un texte, elle va être insérée à l'aide d'un curseur que nous définissons dans la variable `monCurseur`. Nous créons une instance de note vierge attachée à notre document en utilisant la méthode `createInstance` de notre document, avec le service `com.sun.star.text.TextField.Annotation`.

Une note est composée de trois éléments. Tout d'abord, la propriété `Author` est une simple chaîne de caractères contenant le nom de l'auteur. La date de la note (propriété `Date`), est une structure `com.sun.star.util.Date`, que nous remplissons à la date désirée avec une variable intermédiaire. Enfin, le texte de la note est une chaîne de caractères contenue dans la propriété `Content`. Elle peut comporter plusieurs lignes séparées par un caractère « retour de chariot ».

ASTUCE Auteur et date d'une note

Vous pouvez visualiser l'auteur et la date d'une note en ouvrant le Navigateur, section Notes, puis un clic droit sur la note, choisir Notes > Éditer.

Une fois ces renseignements fournis, nous pouvons insérer la note comme tout élément de texte par l'intermédiaire de la méthode `insertString` en utilisant le curseur défini auparavant. La macro que nous donnons en exemple effectue cinq insertions en insérant du texte intermédiaire afin de mieux visualiser le résultat.

Attention : une note compte comme un caractère dans le texte. Restez vigilant lors du déplacement de vos curseurs.

Comme il est possible d'ajouter une note, il est possible de la supprimer par l'intermédiaire de la collection `TextFields`. La macro que nous présentons efface toutes les notes d'un auteur spécifique. Un tri sur la date serait également possible.

```
rem Code11-13.sxw   bibli : ChampsTexte Module3
Option Explicit

Sub EffacerNotesAuteur()
Dim monEnum As Object, uneEnum As Object
Dim monDocument As Object
monDocument = ThisComponent
monEnum = monDocument.TextFields.createEnumeration
while monEnum.hasMoreElements
   uneEnum = monEnum.nextElement
   if uneEnum.supportsService("com.sun.star.text.TextField.Annotation") then
      if uneEnum.Author = "un Auteur" then
         uneEnum.dispose
      endif
   endif
wend
End Sub
```

Nous commençons donc par créer l'énumération des objets `TextField` en appelant la méthode `createEnumeration` de la collection `TextFields`. Nous bouclons ensuite sur tous les éléments. Les objets `TextField` pouvant être autre chose que des notes, nous testons pour chaque élément s'il supporte le service `com.sun.star.text.Textfield.Annotation` par l'intermédiaire de sa méthode `supportsService`.

Si c'est le cas, un test sur sa propriété `Author` nous indique si cette note est candidate à la suppression. Dans l'affirmative, l'appel de la méthode `dispose` de l'objet `TextField` permet de la supprimer.

Textes et paragraphes conditionnels

Plusieurs niveaux de textes conditionnels sont disponibles. Les objets `TextField` proposent l'affichage conditionnel d'une chaîne de caractères ou d'une autre suivant une condition, l'affichage d'un texte sans alternative si une condition est vérifiée, et l'affichage d'un paragraphe complet suivant le même mode.

La macro suivante présente un exemple d'insertion de paragraphe masqué et de champs conditionnels.

```
sub insertMasque()

  leDoc = ThisComponent
  leCurseur = thisComponent.Text.createTextCursor()

  'Paragraphe masqué
  leChamp = leDoc.createInstance( _
        "com.sun.star.text.TextField.HiddenParagraph")
  'la condition associée au paragraphe masqué
  leChamp.condition = "monChamp = ""AA"" "
```

```
    leDoc.Text.insertTextContent(leCurseur, leChamp, false)

    'Champ caché
    leChamp = leDoc.createInstance(_
          "com.sun.star.text.TextField.HiddenText")
    leChamp.condition = "monChamp = ""BB"" "
    leChamp.content =  "Ce texte est caché sous condition"
    leDoc.Text.insertTextContent(leCurseur, leChamp, false)

    'Rafraîchir les champs  - touche F9
    leDoc.TextFields.refresh()

    'Affiche si le champ est masqué ou non
    MsgBox("Le champ est caché : " & cStr(leChamp.isHidden))

end sub
```

Un paragraphe masqué est un champ prédéfini de type `HiddenParagraph`. Nous créons un exemplaire d'un tel objet par la méthode `createInstance` du document sur le service `com.sun.star.text.TextField.HiddenParagraph`.

La propriété `condition` permet de spécifier, à partir d'un champ déjà défini, les cas d'affichage et occultation du paragraphe. La syntaxe est identique à celle employée dans l'interface utilisateur (opérateurs ==, `!=`, `GEQ`, `LEQ`, >, < ...). Nous renvoyons le lecteur à l'aide en ligne de Writer pour le détail de la syntaxe des conditions disponibles.

Dans notre exemple, nous utilisons le champ utilisateur `monChamp` défini précédemment dans ce chapitre. Si `monChamp` est égal à la chaîne de caractère `AA`, le paragraphe contenant ce `TextField` sera masqué. Notez le doublement des guillemets afin de respecter la syntaxe de Basic

Le champ paragraphe masqué peut être inséré comme un `TextField` par la méthode `insertTextContent` de l'objet `Text` du document.

La macro enchaîne ensuite sur l'insertion d'un texte caché conditionnel. Le principe est identique à l'insertion d'un paragraphe masqué. Nous utilisons ici le service `com.sun.star.text.TextField.HiddenText`. La condition se définit de manière identique à l'aide de la propriété `condition`.

Le texte à afficher conditionnellement est défini dans la propriété `content`. Ce texte est une chaîne de caractère et non pas un objet texte au sens de l'API, le formatage à l'intérieur de cette chaîne n'est pas possible. C'est le formatage du champ lui-même qui sera appliqué.

Une fois les différents champs insérés, les `TextFields` du document doivent être rafraîchis par la méthode `refresh`, ce qui permet le calcul des conditions et la mise en page au niveau de l'interface utilisateur.

La méthode `isHidden`, autant pour un `HiddenParagraph` que pour un `HiddenText`, indique par `True` ou `False` si la condition amène à une occultation du champ ou non.

Les signets et renvois

Un signet (anglais : *bookmark*) est un repère dans le texte. Manuellement, vous insérez un signet à l'endroit du curseur visible avec le menu Insertion > Repère de texte... > nomDuSignet. Un signet peut aussi désigner une zone de texte ; il suffit qu'elle soit sélectionnée quand on définit le signet.

Utiliser un signet existant

Vous pouvez placer des signets dans un document pré-défini (ou un modèle de document) pour repérer les endroits que la macro doit remplir. Évidemment, la macro doit connaître le nom de chaque signet et le type d'information à y insérer.

L'exemple suivant initialise le curseur d'écriture à l'emplacement repéré par un signet, puis écrit un texte. Le document du zip téléchargeable contient un signet dans le texte principal, un autre dans un tableau, un troisième désignant une zone dans l'en-tête.

```
rem Code11-10.sxw   bibli : EcrireSurSignet Module1
Option Explicit

Sub InsererAuSignet()
Dim monDocument As Object, leTexte As Object
Dim unSignet As Object, monCurseur As Object
monDocument = ThisComponent
' signet dans le texte principal
unSignet = monDocument.Bookmarks.getByName("repère1")
leTexte = unSignet.Anchor.Text
monCurseur = leTexte.createTextCursorByRange(_
                        unSignet.Anchor.Start)
monCurseur.CharPosture = com.sun.star.awt.FontSlant.ITALIC
leTexte.insertString( monCurseur, "la ville de ", false)
' signet dans une cellule d'un tableau Writer
unSignet = monDocument.Bookmarks.getByName("repère2")
leTexte = unSignet.Anchor.Text
monCurseur = leTexte.createTextCursorByRange(_
                        unSignet.Anchor.Start)
leTexte.insertString( monCurseur, " derrière", false)
unSignet = monDocument.Bookmarks.getByName("repère3")
print "Repère3 = " & unSignet.Anchor.String
End Sub
```

Nous récupérons d'abord le signet parmi la collection des signets du document, ensuite la propriété `Text` nous fournit le support de texte dans lequel le signet est inséré. La propriété `Anchor` (ancre) du signet est une zone de texte à partir de laquelle nous créons un curseur d'écriture. L'intérêt de cette méthode est de fonctionner quel que soit le support texte dans lequel se trouve le signet, par exemple dans un tableau ou dans un bas de page. Nous récupérons le texte repéré par le troisième signet grâce à la propriété `String` de l'objet `Anchor`.

Après avoir essayé l'exemple sur un document, si vous annulez les modifications de texte, vous constaterez que les signets disparaîtront aussi !

Le nom du signet doit être écrit exactement comme dans sa définition. L'instruction `getByName` déclenchera une exception s'il n'existe aucun signet de ce nom. Vous pouvez tester l'existence du signet ainsi :

```
if monDocument.Bookmarks.hasByName("ecrire_ici") then
  rem le signet existe
end if
```

L'objet `Anchor` peut fournir l'objet dans lequel se trouve le signet, par exemple une cellule de tableau ou un cadre, voir tableau 11-48.

Tableau 11–48 Sites possibles d'implantation d'un signet

Propriété	Type	Site d'implantation du signet
`Cell`	`Object`	Une cellule d'un tableau.
`TextFrame`	`Object`	Un cadre.
`TextTable`	`Object`	Le tableau auquel appartient la cellule.
`TextSection`	`Object`	Une section de texte.
`TextField`	`Object`	Un champ.
`FootNote`	`Object`	Une note de bas de page.
`EndNote`	`Object`	Une note de fin de document.

Ces différents objets peuvent être non significatifs pour un signet particulier. On vérifie facilement l'existence d'un objet avec la fonction Basic `IsEmpty` :

```
maCellule = unSignet.Anchor.Cell
if IsEmpty(maCellule) then
  print "Le signet n'est pas dans une cellule"
else
  ' le signet est bien dans une cellule
end if
```

Un signet peut s'avérer utile pour renvoyer contextuellement par macro vers la page le contenant. S'agissant d'affichage, nous allons tout naturellement nous tourner vers la notion de curseur visible vue précédemment.

```
rem Code11-10.sxw   bibli : EcrireSurSignet Module4
Option Explicit

Sub RenvoyerAuSignet()
Dim monDocument As Object, curseurVisible As Object
Dim unSignet As Object, laCible As Object
monDocument = ThisComponent
unSignet = monDocument.Bookmarks.getByName("repère3")
laCible = unSignet.Anchor
curseurVisible = monDocument.CurrentController.ViewCursor
curseurVisible.gotoRange(laCible,false)
End Sub
```

À partir du nom du signet, nous pouvons utiliser la méthode `getByName` de la collection `Bookmarks` du document pour récupérer le signet en question.

L'objet `ViewCursor` a une méthode `gotoRange` prenant un `TextRange` comme argument. Nous créons donc la variable `laCible` contenant l'objet `Anchor` de notre signet. Cette variable `laCible` est un `TextRange`.

Il ne nous reste plus alors qu'à obtenir le curseur visible et appeler sa méthode `gotoRange`.

Insérer un signet

L'exemple suivant insère un signet dans le texte principal.

```
rem Code11-10.sxw   bibli : EcrireSurSignet Module2
Option Explicit

Sub AjouterUnSignet()
Dim monDocument As Object, monTexte As Object
Dim monSignet As Object, monCurseur As Object
monDocument = ThisComponent
monSignet = monDocument.createInstance(_
                        "com.sun.star.text.Bookmark")
monSignet.Name = "Signet_ajouté"
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False)
monCurseur.gotoNextSentence(False)
monTexte.insertTextContent(monCurseur, monSignet, false)
End Sub
```

Trouver les signets

L'objet `Bookmarks` nous permet de connaître tous les signets du document. Le nombre de signets est fourni par la propriété `Count`, leurs noms sont obtenus avec la propriété `Name` de chaque objet signet.

```
rem Code11-10.sxw   bibli : EcrireSurSignet Module3
Option Explicit

Sub ListerSignets()
Dim monDocument As Object, leTexte As Object
Dim unSignet As Object, lesSignets As Object
Dim x As Long
monDocument = ThisComponent
lesSignets = monDocument.Bookmarks
for x = 0 to lesSignets.Count -1
  unSignet= lesSignets(x)
  print "Signet : " & unSignet.Name
next
End Sub
```

Attention : l'ordre des signets dans la liste de `Bookmarks` ne correspond pas systématiquement à l'ordre dans le document.

Liens hypertextes

Les gros documents ou les documents scientifiques peuvent tirer avantage de facilités de navigation procurés par les liens hypertextes. Ces liens peuvent être de plusieurs natures : interne au document, en direction d'un autre document ou même directement sur Internet.

Bien que plusieurs méthodes soient possibles, nous présentons celle que nous considérons la plus simple.

```
monCurseur = thisComponent.Text.createTextCursor
monCurseur.HyperlinkURL = "#Tableau1|table"
moncurseur.String = "Essai"
```

Cette méthode est basée sur un `TextCursor` dont il suffit de définir la propriété `HyperlinkURL`. Avec ce qui a été vu auparavant, une plage quelconque de caractères peut être ainsi définie comme lien hypertexte.

Maintenant, nous avons trois possibilités de liens :

- lien interne au document,
- lien vers un autre document,
- lien externe (URL).

La différence ne va se faire que sur le contenu de la propriété `HyperlinkURL`.

Lien externe

Si vous avez défini un navigateur par défaut dans le paramétrage d'OpenOffice.org (Outils > Options > OpenOffice.org > Programmes auxiliaires), vous pouvez définir un lien hypertexte pointant vers Internet dans votre document. Il suffit de définir par exemple :

```
monCurseur.HyperlinkURL = "http://www.google.fr"
```

Cette URL peut également contenir des arguments pour les soumettre à un script sur un serveur.

Lien vers un autre document

Pour que le lien pointe sur un autre document, il suffit d'indiquer le chemin de ce document sous forme d'URL :

```
monCurseur.HyperlinkURL = ConvertToURL("C:\Fichier.sxw")
```

Lien interne au document

Il est possible de positionner l'arrivée du lien sur un élément du document, en employant son nom. OpenOffice.org permet de nommer les objets inclus dans les documents. Le Navigateur accessible par la touche F5 permet de connaître tous les éléments du document classés par type.

Cette notion de type est importante. Deux objets de nature différente pouvant porter le même nom, le lien devra contenir cette information.

En faisant une insertion manuelle de lien hypertexte interne, nous constatons, pour un tableau par exemple, que la syntaxe est la suivante :

```
monCurseur.HyperlinkURL = "#Tableau1|table"
```

Nous en déduisons donc la règle suivante pour construire les liens internes :

```
monCurseur.HyperlinkURL = "#" & NomObjet &"|" & TypeObjet
```

Le contenu de la variable `TypeObjet` peut être retrouvé lors d'une insertion manuelle. Le tableau 11-49 donne les principaux types d'objets.

Tableau 11–49 Types d'objets des liens hypertextes

Objet	NomObjet	TypeObjet	Remarques
Tableau	Le nom du tableau.	`|table`	
Image	Le nom de l'image.	`|graphic`	
Cadre	Le nom du cadre.	`|frame`	
Section	Le nom de la section.	`|region`	
Signet	Le nom du signet		Pas d'extension TypeObjet, même si le signet se trouve dans un tableau, dans un cadre, etc.
Titre	La chaîne de caractères du titre.	`|outline`	
Objet OLE	Le nom de l'objet.	`|ole`	L'objet a été inséré par exemple avec le menu Insertion > Objet

Lien vers un endroit dans un autre document

Enfin pour terminer, en combinant les deux méthodes précédentes, il est possible de construire un lien hypertexte sur un fichier externe à un endroit spécifique :

```
monURL = ConvertToURL("C:\Fichier.sxw")
monCurseur.HyperlinkURL = monURL & "#tableau1|table"
```

Configuration d'affichage du document

Ces propriétés du document Writer (tableau 11-50), qui existent dans l'interface utilisateur principalement au niveau Outils>Options>Texte>Affichage et aide au formatage, sont accessibles à partir de l'objet contrôleur du document. Elles ne sont pas mémorisées à la sauvegarde du document.

```
rem Code11-08.sxw   bibli : Config Module1
Option Explicit

Sub ConfigDocWriter()
Dim monDocument As Object, confVisu As Object
monDocument = thisComponent
confVisu = monDocument.CurrentController.ViewSettings
With confVisu
  .ShowVertRuler = MsgBox("ShowVertRuler ?", 4) = 6
  .IsVertRulerRightAligned = MsgBox("IsVertRulerRightAligned ?", 4) = 6
  .ShowOnlineLayout = MsgBox("ShowOnlineLayout ?", 4) = 6
end With
End Sub
```

Tableau 11–50 Propriétés de configuration de Writer

Propriété	Type	Signification
ShowAnnotations	Boolean	True pour afficher l'indicateur de note.
ShowBreaks	Boolean	True pour afficher les retours de ligne.
ShowParaBreaks	Boolean	True pour afficher les fins de paragraphe.
ShowFootnoteBackground	Boolean	True pour afficher un fond gris sur les symboles de note de bas de page.
ShowIndexMarkBackground	Boolean	True pour afficher un fond gris sur les marques d'index.
ShowProtectedSpaces	Boolean	True pour afficher un fond gris sur les espaces insécables.
ShowSoftHyphens	Boolean	True pour afficher un fond gris sur les tirets conditionnels.
ShowSpaces	Boolean	True pour afficher un point pour chaque espace.
ShowTabstops	Boolean	True pour afficher les tabulations.
ShowTextFieldBackground	Boolean	True pour afficher un fond gris sur les champs.
ShowHiddenParagraphs	Boolean	True pour afficher les paragraphes masqués.
ShowHiddenText	Boolean	True pour afficher le texte caché.
ShowRulers	Boolean	True pour autoriser l'affichage des règles.
ShowHoriRuler	Boolean	True pour afficher la règle horizontale, si ShowRulers vaut True.
ShowVertiRuler	Boolean	True pour afficher la règle verticale, si ShowRulers vaut True.
ShowTableBoundaries	Boolean	True pour afficher les limites des tableaux.
ShowTables	Boolean	True pour afficher les tableaux.
ShowHoriScrollBar	Boolean	True pour afficher l'ascenseur horizontal.
ShowVertScrollBar	Boolean	True pour afficher l'ascenseur vertical.
IsVertRulerRightAligned	Boolean	True pour afficher les positions de la règle verticale et de l'ascenseur vertical.
SmoothScrolling	Boolean	True pour un défilement doux.
SolidMarkHandles	Boolean	True pour afficher des grandes poignées de déplacement.
ShowGraphics	Boolean	True pour afficher les images.
ShowDrawings	Boolean	True pour afficher les formes.
ShowFieldCommands	Boolean	True pour afficher le contenu des champs.
ShowOnlineLayout	Boolean	True pour afficher comme un document HTML.
ZoomType	Integer	Facteur de zoom ; constante nommée, voir chapitre 10.
ZoomValue	Integer	Valeur du zoom, voir chapitre 10.

Conclusion

Nous venons de parcourir les différents concepts accessibles par l'API et concernant les documents Writer. Notre approche orientée vers la rédaction, les notions de curseur et de style y ont été exposées.

Le chapitre suivant traitera de l'API pour manipuler un document Calc.

12

Les documents Calc

Ce chapitre montre comment accéder aux objets élémentaires de Calc que sont les feuilles et les cellules ; il décrit leurs propriétés et explique comment les manipuler. Nous verrons également des méthodes qui répondent aux besoins les plus courants : activer le calcul des formules, lire et écrire les données d'un tableau, invoquer une fonction de Calc, insérer un lien hypertexte, etc. D'autres manipulations plus spécialisées, notamment sur les styles, les zones nommées, les en-têtes, pieds de page et diagrammes, sont aussi présentées avec des exemples. Enfin, l'impression d'un document Calc présente quelques particularités que nous décrirons.

> API **Référence sur Calc (en anglais)**
>
> La documentation de l'API est décrite dans le chapitre 8 du « Developer's Guide » du SDK. Elle comporte de nombreux liens hypertextes vers les descriptions IDL du SDK.

Lecture et manipulation de feuilles

Accéder aux feuilles existantes

L'objet `Sheets` (anglais pour feuilles) représente l'ensemble des feuilles d'un document Calc. Le nombre actuel de feuilles est exposé par la propriété `Count` de l'objet `Sheets`.

Chaque feuille du document Calc est elle-même un objet. Basic nous permet d'obtenir facilement une des feuilles en considérant l'objet `Sheets` comme un tableau de feuilles dont l'index varie de zéro au nombre de feuilles moins un. L'ordre des feuilles dans le document est reflété par la valeur de l'index. Le nom d'une feuille nous est indiqué dans sa propriété `Name`.

Nous en savons suffisamment pour énumérer les feuilles d'un document Calc :

```
rem Code12-01.sxc   bibli : Feuilles Module1
Option Explicit

Sub EnumererFeuilles()
Dim monDocument As Object, lesFeuilles As Object
Dim uneFeuille As Object
Dim x As Long, nbF As Long

monDocument = thisComponent
lesFeuilles = monDocument.Sheets
nbF = lesFeuilles.Count
print "Nombre de feuilles : " & nbF
for x = 0 to nbF -1
  uneFeuille = lesFeuilles(x)
  print "Feuille : " & uneFeuille.Name
next
End Sub
```

Un classeur Calc, dans sa version francisée, nomme les feuilles Feuille1, Feuille2, Feuille3, etc. Si le classeur a été réalisé avec une version localisée dans une autre langue, il comportera par défaut d'autres noms de feuilles, par exemple en version anglaise Sheet1, Sheet2, etc. Nous accéderons aux feuilles d'un classeur avec les index 0, 1, 2, etc. Cependant, comme l'utilisateur peut changer l'ordre des feuilles, vous ne devez supposer aucune correspondance entre index et identité de feuille. Le seul moyen sûr d'obtenir l'objet feuille que l'on souhaite consiste à nommer la feuille puis utiliser son nom. L'objet `Sheets` nous fournit pour cela la méthode `getByName`, qui prend en argument le nom de la feuille souhaitée et renvoie l'objet correspondant, s'il existe. S'il n'existe pas, une erreur se produira ; aussi l'objet `Sheets` nous donne-t-il le moyen de tester l'existence d'une feuille d'un nom donné avec la fonction `hasByName`, qui renvoie `True` dans le cas positif.

Nous allons utiliser ces notions pour renommer une feuille. Pour cela, il suffit de modifier la propriété `Name` de la feuille. Après quoi, nous exécuterons la macro précédente pour lister les différentes feuilles.

```
rem Code12-01.sxc   bibli : Feuilles Module2
Option Explicit

Sub RenommerFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim uneFeuille As Object
Dim nom1 As String, nom2 As String
```

```
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

nom1 = InputBox("Nom actuel de la feuille")
if lesFeuilles.hasByName(nom1) then
  nom2 = InputBox("Nouveau nom pour la feuille")
  ' récupérer la feuille "nom1"
  uneFeuille = lesFeuilles.getByName(nom1)
  uneFeuille.Name = nom2 ' renommer cette feuille
  EnumererFeuilles  ' lister les feuilles du document
else
  MsgBox(nom1 & " n'existe pas",16)
end if
End Sub
```

Exécutez la macro, vous verrez l'onglet de la feuille concernée changer de nom. Attention à la casse ! Les noms de feuilles doivent être écrits en respectant les majuscules et minuscules.

Ajouter une nouvelle feuille

La méthode `insertNewByName` de l'objet `Sheets` sert à créer une nouvelle feuille vierge.

- Le premier argument est le nom de la nouvelle feuille.
- Le deuxième est la position que celle-ci occupera dans le classeur, décalant les feuilles existantes à partir de cette position.

Si la position a pour valeur zéro, la feuille sera insérée en tête. Si la valeur de position est hors limites, par exemple négative, la feuille sera ajoutée à la fin.

Il n'est pas vraiment pratique d'insérer une feuille à une position, car l'utilisateur peut modifier la structure du document. Il est plus sûr de se positionner par rapport à une feuille dont on connaît le nom. Comment récupérer la position de cette feuille ? Anticipant sur la section « Obtenir les coordonnées d'une zone de cellule », nous utilisons la propriété `RangeAddress` de la feuille. Elle fournit une structure dont l'élément `Sheet` est précisément la position de la feuille.

```
rem Code12-01.sxc   bibli : Feuilles Module3
Option Explicit

Sub AjouterFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim F1 As Object, indexF1 As Long
Dim nom1 As String, nom2 As String
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

nom1 = InputBox("Insérer après la feuille :")
if lesFeuilles.hasByName(nom1)  then
  nom2 = InputBox("La nouvelle feuille aura pour nom :")
```

```
  F1 = lesFeuilles.getByName(nom1)
  indexF1 = F1.RangeAddress.Sheet
  ' insérer après la feuille "nom1"
  lesFeuilles.insertNewByName(nom2, indexF1 +1)
else
  MsgBox(nom1 & " n'existe pas", 16)
end if
End Sub
```

Supprimer une feuille

La méthode `removeByName` de l'objet `sheets` permet de supprimer une feuille à partir de son nom. Cet exemple demande un nom de feuille à supprimer ; si la chaîne de caractères est nulle, c'est que la demande est annulée ; dans le cas contraire, nous vérifions qu'une feuille de ce nom existe avant de la supprimer.

```
rem Code12-01.sxc   bibli : Feuilles Module4
Option Explicit

Sub SupprimerFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim nom1 As String
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

Do
  nom1 = InputBox("Feuille à supprimer ?")
  if nom1 = "" then exit sub ' annulation, ne rien supprimer
  ' reposer la question en cas d'erreur
Loop Until lesFeuilles.hasByName(nom1)
lesFeuilles.removeByName(nom1) ' supprimer la feuille
End Sub
```

Dupliquer une feuille

La méthode `insertNewByName` ajoute une feuille vierge. Si votre classeur comporte une feuille parfaitement formatée et remplie, il est avantageux de créer une feuille en faisant une copie de la première. C'est l'objet de la méthode `copyByName`, qui comporte trois arguments, successivement :

- le nom de la feuille servant de modèle,
- le nom de la nouvelle feuille,
- la position que celle-ci occupera dans le classeur, décalant les feuilles existantes à partir de cette position.

Dans cet exemple, nous ajoutons la feuille `Février` après la feuille `Janvier`, en copiant celle-ci. Pour la placer à la bonne position, nous utilisons le même principe que pour ajouter une feuille.

```
rem Code12-01.sxc   bibli : Feuilles Module5
Option Explicit

Sub DupliquerFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim F1 As Object, indexF1 As Long
Dim nom1 As String, nom2 As String
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

nom1 = "Janvier"
nom2 = "Février"
F1 = lesFeuilles.getByName(nom1)
indexF1 = F1.RangeAddress.Sheet
' créer la feuille Février à l'image de Janvier
lesFeuilles.copyByName(nom1, nom2, indexF1 +1)
End Sub
```

Déplacer une feuille dans le classeur

La méthode `moveByName` de l'objet `Sheets` permet de déplacer une feuille à une autre position dans la série de feuilles du classeur. Elle prend comme arguments le nom de la feuille à déplacer et la nouvelle position. Par exemple, ce codage place `Feuille3` en début de classeur.

```
rem Code12-01.sxc   bibli : Feuilles Module6
Option Explicit

Sub DeplacerFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim uneFeuille As Object
monDocument = thisComponent

lesFeuilles = monDocument.Sheets
lesFeuilles.moveByName("Feuille3", 0)
End Sub
```

La feuille visible par l'utilisateur

La feuille visible dans l'interface utilisateur est appelée en anglais *active sheet* (feuille active). Nous pouvons la récupérer avec la propriété `activeSheet` de l'objet contrôleur associé au document. Pour rendre visible une autre feuille, il suffit de l'affecter à la propriété `activeSheet`.

```
rem Code12-01.sxc   bibli : Feuilles Module7
Option Explicit

Sub FeuilleVisible()
Dim monDocument As Object, lesFeuilles As Object
Dim uneFeuille As Object
Dim texte1 As String, nom2 As String
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

uneFeuille = monDocument.currentController.activeSheet
texte1 = "Feuille active : " & uneFeuille.Name & chr(13)
nom2 = InputBox(texte1 & "Quelle feuille rendre active ?")
if lesFeuilles.hasByName(nom2) then
  uneFeuille = lesFeuilles.getByName(nom2)
  monDocument.currentController.activeSheet = uneFeuille
end if
End Sub
```

Protéger une feuille

La protection d'une feuille consiste seulement à lui affecter un mot de passe. Le niveau de protection est défini au niveau des cellules, nous le verrons dans la section correspondante.

Un objet feuille dispose de trois méthodes :

- `isProtected` est une fonction booléenne qui renvoie `True` si la feuille est protégée par un mot de passe.
- `protect` applique le mot de passe donné en argument (une chaîne de caractères).
- `unprotect` supprime le mot de passe, à condition qu'il soit donné en argument.

Vous remarquerez qu'il n'existe pas de méthode permettant de lire le mot de passe. Pour l'exemple, nous appliquerons seulement un mot de passe statique. En pratique, le mot de passe doit être fourni par l'utilisateur.

```
rem Code12-01.sxc   bibli : Feuilles Module8
Option Explicit

Sub ProtectionFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, motPasse As String
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Contenus")
if maFeuille.isProtected then
  motPasse = InputBox("La feuille est déjà protégée." & _
             chr(13) & "Mot de passe ?")
  maFeuille.unprotect(motPasse) ' enlever le mot de passe
```

```
elseif MsgBox("Protéger la feuille ?", 260) = 6 then
  maFeuille.protect("OpenOffice") ' mettre un mot de passe
end if
End Sub
```

Cellules et zones de cellules

Une feuille de tableau se compose de cellules, auxquelles nous pouvons accéder individuellement. Pour certains usages, on définit une zone de cellules, représentant un rectangle composé de cellules.

Obtenir une cellule

L'objet feuille fournit trois méthodes pour faire référence à une cellule et y accéder :

- en utilisant les coordonnées alphanumériques habituelles d'une cellule,
- en utilisant le *nom* d'une zone réduite à cette cellule (menu Insertion > Noms > Définir...),
- en utilisant des coordonnées X, Y pour lesquelles X est le rang de la colonne et Y est le rang de la ligne, tous deux numérotés à partir de zéro.

Dans l'exemple qui suit, nous récupérons de trois manières différentes la cellule de coordonnées `C2` dans la feuille nommée `Janvier`. Nous avons défini auparavant dans la feuille le nom de zone `CelluleC2`. La propriété `Value` de l'objet cellule renvoie la valeur numérique du contenu d'une cellule ; nous l'utilisons pour montrer que nous avons bien récupéré la bonne cellule.

```
rem Code12-01.sxc   bibli : ZonesCellules Module1
Option Explicit

Sub TrouverCellule()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Janvier")

' trois manières d'obtenir la cellule de coordonnées C2
maCellule = maFeuille.getCellRangeByName("C2")
print "Accès alphanumérique = " & maCellule.value
maCellule = maFeuille.getCellRangeByName("CelluleC2")
print "Accès par le nom = " & maCellule.value
maCellule = maFeuille.getCellByPosition(2,1)
print "Accès par la position XY = " & maCellule.value
End Sub
```

La zone nommée doit concerner la même feuille que `maFeuille` dans l'exemple.

Attention à la casse ! Les noms de zone doivent être écrits en respectant les majuscules et minuscules.

Obtenir une zone de cellules

On récupère un objet zone de cellules de manière très similaire :

```
maZone = uneFeuille.getCellRangeByName("B2:AF10")
maZone = uneFeuille.getCellRangeByName("zoneValeurs")
maZone = uneFeuille.getCellRangeByPosition(1,1, 31,9)
```

Dans la première manière, vous reconnaissez la définition d'une zone dans Calc. La deuxième utilise une zone nommée située dans la feuille. La troisième manière utilise une nouvelle méthode qui prend quatre arguments : les coordonnées colonne et ligne (X et Y) de deux cellules de coin de la zone, situées en diagonale.

> Notez qu'une zone de plusieurs cellules est un type d'objet différent d'un objet cellule, bien que les deux aient nombre de propriétés identiques. En revanche, une zone d'une seule cellule *est* un objet cellule.

Nous pouvons parfaitement récupérer une cellule en se repérant en relatif par rapport à l'intérieur d'une zone de cellules. Il suffit d'utiliser la méthode `getCellByPosition` de l'objet zone de cellules.

```
rem Code12-01.sxc   bibli : ZonesCellules Module2
Option Explicit

Sub TrouverCelluledeZone()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim maZone As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Janvier")

' trois manières d'obtenir la même zone
'maZone = maFeuille.getCellRangeByName("B2:AF10")
'maZone = maFeuille.getCellRangeByName("zoneValeurs")
maZone = maFeuille.getCellRangeByPosition(1,1, 31,9)
' récupérer la cellule C2
maCellule = maZone.getCellByPosition(1,0)
print "Accès par position XY d'une zone = " & maCellule.value
End Sub
```

La cellule de coordonnées C2 est située sur la deuxième colonne et la première ligne de la zone B2:AF10, donc ses coordonnées XY relatives à la zone sont 1 et 0. De la même manière, on pourrait définir une autre zone de cellules à l'intérieur de la première zone, en utilisant getCellRangeByPosition avec des coordonnées XY relatives.

À l'inverse, getCellRangeByName utilise toujours des coordonnées absolues, même si vous utilisez cette méthode à partir d'un objet zone.

Obtenir les coordonnées d'une cellule

Disposant d'une cellule quelconque, par exemple obtenue dynamiquement, on obtient ses coordonnées absolues grâce à sa propriété CellAddress, qui se compose de trois éléments :

- Sheet de type Integer, rang de la feuille dans le classeur, numéroté à partir de zéro,
- Column et Row, de type Long, qui sont respectivement les coordonnées X et Y (c'est-à-dire colonne et ligne) par rapport à la feuille.

Nous avons précédemment indiqué comment obtenir une feuille à partir de son rang, puis obtenir le nom de la feuille. Cependant, nous pouvons obtenir plus directement l'objet feuille dans lequel se trouve la cellule :

```
uneFeuille = maCellule.Spreadsheet
```

Certaines méthodes de l'API utilisent en argument un objet coordonnées de cellule. Il se définit soit à partir d'une cellule et de sa propriété CellAddress, soit directement :

```
Dim cooCell As New com.sun.star.table.CellAddress

With cooCell  ' cellule feuille2.H14
  .Sheet  = 1 ' deuxième feuille du document tableur
  .Column = 7 ' colonne H
  .Row    = 13 ' ligne 14
End With
```

L'annexe B offre la routine utilitaire alphaXY, qui sert à convertir une coordonnée XY en une adresse alphanumérique classique, et la routine adresseString, qui convertit une adresse de type Celladdress en adresse sous forme de texte.

Obtenir les coordonnées d'une zone de cellule

De manière similaire, une zone de cellules nous donne ses coordonnées absolues grâce à sa propriété RangeAddress, qui se compose de cinq éléments :

- Sheet de type Integer, rang de la feuille dans le classeur, numéroté à partir de zéro,
- StartColumn et StartRow, de type Long, qui sont les coordonnées X et Y de début de la zone,

- `EndColumn` et `EndRow`, de type `Long`, qui sont les coordonnées X et Y de fin de la zone.

L'objet feuille dans lequel se trouve la zone est obtenu directement :

```
uneFeuille = maZone.Spreadsheet
```

Un objet coordonnées de zone de cellules peut aussi être défini directement :

```
Dim cooZone As New com.sun.star.table.CellRangeAddress

With cooZone        ' zone feuille2.H14:L23
  .Sheet  = 1       ' deuxième feuille du document tableur
  .StartColumn = 7 ' colonne H
  .StartRow   = 13 ' ligne 14
  .EndColumn =  11 ' colonne L
  .EndRow   =   22 ' ligne 23
End With
```

La routine `adrZoneString` de l'annexe B convertit une adresse de type `RangeAddress` en adresse sous forme de texte.

Nous utiliserons ces notions avec les sélections visuelles.

Les zones nommées

Dans la macro qui suit, nous allons récupérer la liste des zones nommées d'un document Calc, ajouter une zone nommée puis la supprimer. Les zones nommées sont contenues dans l'objet `NamedRanges` du document. Le document du zip téléchargeable contient plusieurs zones nommées. En Basic, on accède à chaque zone par une simple indexation de cet objet, qui fournit le nombre de zones dans sa propriété `Count`. D'autres explications sont données après la macro.

```
rem Code12-01.sxc   bibli : ZonesCellules Module5
Option Explicit

Sub NommerZone()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim lesZonesNom As Object, maZoneNom As Object, x As Long
Const unNom = "Genre de dépense"
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Janvier")
maCellule = maFeuille.getCellRangeByName("C2")
lesZonesNom = monDocument.NamedRanges
for x = 0 to lesZonesNom.Count -1
  maZoneNom = lesZonesNom(x)
  print "Zone n°" & x , maZoneNom.Name, "", maZoneNom.Content
next
```

```
if not lesZonesNom.hasByName(unNom) then
  lesZonesNom.addNewByName(unNom, "Janvier.A2:A7", _
              maCellule.CellAddress,0)
  print "Nouvelle zone nommée : " & unNom
end if
if MsgBox("Effacer l'exemple " & unNom & " ?", 4) = 6 then
  lesZonesNom.removeByName(unNom)
end if
End Sub
```

Chaque zone comporte deux propriétés de type `String` :

- `Name` contient le nom de la zone.
- `Content` contient dans notre exemple une adresse de zone, mais pourrait aussi bien contenir une formule avec des adresses ; les adresses peuvent être absolues, relatives, ou partiellement relatives.

Avant d'ajouter un nom de zone, nous devons nous assurer qu'il n'existe pas déjà, grâce à la fonction `hasByName` de l'objet `lesZonesNom`. La méthode `addNewByName`, du même objet, utilise quatre paramètres :

1. le nom de la nouvelle zone,
2. la définition de la zone, ici une simple adresse, mais éventuellement une formule,
3. une adresse de cellule servant de référence, qui doit se situer dans la même feuille,
4. le type de la zone, zéro dans la plupart des cas.

Pourquoi une adresse de cellule de référence ? Nous pouvons très bien utiliser une zone nommée dans une formule d'une cellule quelconque, ou la zone peut être elle-même une formule qui sera stockée dans une cellule. Dans ces cas, si la définition contient des adresses relatives, elles seront mises à jour en fonction des positions relatives. Dans la feuille `Janvier`, nous avons mis la formule `=SOMME(GenreDepense)` dans la cellule `C16`. Tant que la zone de ce nom n'est pas définie, la cellule affiche une erreur. Quand la zone est définie par la macro, elle affiche la somme des cellules `A16:A20`.

Enfin, l'objet `lesZonesNom` offre la méthode `removeByName`, qui nous permet de supprimer une zone nommée.

Pour plus d'information sur les zones nommées, consulter le chapitre 8.3.3 du « Developer's Guide » et les interfaces `XNamedRanges` et `XnamedRange` dans le SDK.

Les sélections visuelles

Sélection faite par l'utilisateur

L'utilisateur peut sélectionner une cellule, une zone de cellules, ou même plusieurs zones, puis appeler une macro qui fera un traitement en fonction de ces zones. L'objet

`currentSelection` obtenu du document nous donne des informations sur la sélection, mais d'une manière complexe car la nature de cet objet est différente dans chaque cas.

La distinction se fera en recherchant quels services sont supportés par `currentSelection`, grâce à sa fonction `supportsService`, qui renvoie `True` si le service en argument est reconnu. La macro suivante, que nous allons expliquer, affiche les coordonnées de chaque zone sélectionnée. Nous utilisons des routines de l'annexe B qui sont recopiées dans le module `Utilitaires` de la bibliothèque `Standard` du même document.

```
rem Code12-01.sxc   bibli : ZonesCellules Module3
Option Explicit

Sub AfficherCoordonneesSelection()
Dim monDocument As Object, lesFeuilles As Object
Dim sel As Object, coord As Object
Dim x As Long, zonex As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
sel = monDocument.currentSelection
if sel.supportsService(_
       "com.sun.star.sheet.SheetCellRanges") then
  for x = 0 to sel.Count -1 ' balayer les zones
    zonex = sel(x)
    MsgBox "Zone n°" & x & " =  " & _
           adrZoneString(monDocument, zonex.RangeAddress)
  next
elseif sel.supportsService(_
       "com.sun.star.table.Cell") then ' une seule cellule
  coord = sel.CellAddress
  MsgBox "Une cellule =   " & _
         adresseString(monDocument, coord)
elseif sel.supportsService(_
       "com.sun.star.table.CellRange") then ' une seule zone
  MsgBox "Une zone =  " & _
         adrZoneString(monDocument, sel.RangeAddress)
else
  MsgBox("Erreur logicielle !", 16)
end if
End Sub
```

Les tests successifs sur les services reconnus doivent être faits dans l'ordre indiqué, car une seule cellule reconnaît aussi le service `CellRange`.

Attention à la casse ! Les noms de services doivent être écrits en respectant les majuscules et minuscules.

Si plusieurs zones sont sélectionnées, l'objet obtenu par `currentSelection` fournit leur nombre avec sa propriété `Count`. En Basic, une simple indexation de l'objet nous donne les objets zones de cellules. En utilisant les propriétés de coordonnées de zone décrites

plus haut, et à l'aide de la routine `adrZoneString` (voir l'annexe B), nous affichons les coordonnées de chaque zone.

Si nous n'avons qu'une seule zone, celle-ci est directement disponible. Si une cellule seulement est sélectionnée (ou si simplement le curseur est sur une cellule), la cellule est directement disponible et nous affichons ses coordonnées avec la routine `adresseString` de l'annexe B.

Cette macro nous donne l'occasion de faire deux manipulations intéressantes :

- Cliquez sur la feuille dans la case en coin haut-gauche des coordonnées, pour sélectionner toute la feuille ; affichez les coordonnées.
- Sélectionnez un grand rectangle de cellules. Avec la touche Ctrl appuyée, cliquez sur quelques cellules au hasard : votre sélection est maintenant « trouée ». Affichez les coordonnées : vous constaterez que votre sélection est définie par un nombre de zones rectangulaires suffisant pour la couvrir.

Afficher une zone sélectionnée

Faire afficher dans la fenêtre Calc une zone de cellules sélectionnée est bien plus simple.

```
rem Code12-01.sxc   bibli : ZonesCellules Module4
Option Explicit

Sub AfficherZoneSelectionnee()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim maZone As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Janvier")
maZone = maFeuille.getCellRangeByName("E5:F7")
monDocument.currentController.Select(maZone)
End Sub
```

Si le tableur affichait une autre feuille, celle contenant la sélection deviendrait visible.

Zone visible dans la feuille

Figer des lignes ou colonnes

Le menu `Fenêtre>Fixer` permet de maintenir la vision des en-têtes d'un tableau dépassant la taille de la feuille. Ceci est réalisé par programmation avec la méthode `freezeAtPosition` de l'objet contrôleur du tableur ; elle prend en argument le nombre de colonnes et de lignes à figer. Pour débloquer ce type d'affichage, il est nécessaire que la cellule active soit en position A1.

```
rem Code12-01.sxc   bibli : Feuilles Module9
Option Explicit

Sub FigerLignesColonnes()
Dim monControleur As Object, maFeuille As Object, position As Object
monControleur = ThisComponent.CurrentController
if MsgBox("Figer 3 lignes et 2 colonnes ?", 4) = 6  then
 monControleur.freezeAtPosition(3,2)
else
  maFeuille = monControleur.ActiveSheet
  position = maFeuille.getCellRangeByName("A1")
  monControleur.select(position)
  monControleur.freezeAtPosition(0,0)
end if
End Sub
```

Première ligne et première colonne affichées

Sur une feuille trop grande pour être visualisée en totalité, vous pouvez choisir les coordonnées de la cellule qui sera affichée dans le coin gauche, en haut.

```
Dim monDocument As Object, monControleur As Object
monDocument = ThisComponent
monControleur = monDocument.CurrentController
' choisir la feuille à afficher - ici la deuxième
monControleur.ActiveSheet = monDocument.Sheets(1)
monControleur.FirstVisibleColumn = 17   ' colonne R
monControleur.FirstVisibleRow = 153     ' ligne 154
```

Inversement, vous pouvez connaître les coordonnées de la première cellule affichée en lisant la valeur de ces deux propriétés. De plus, l'objet contrôleur expose la propriété `VisibleRange`, qui est une structure de type `com.sun.star.table.CellRangeAddress` déjà vue à la section « Obtenir les coordonnées d'une cellule ». Cette propriété, en lecture seule, vous donne les coordonnées extrêmes de l'ensemble des cellules affichées.

Propriétés globales de la cellule

Ces propriétés concernent l'aspect de la cellule ; les modifier n'entraîne pas une modification du style de la cellule. De plus, une interrogation du style de la cellule donnera toujours le même nom alors que la cellule aura changé d'aspect. Une zone de cellules possède les mêmes propriétés, ce qui permet de modifier toutes les cellules de la zone de manière identique.

Le tableau 12-1 montre la liste des propriétés simples concernant l'aspect d'une cellule. Nous verrons ensuite séparément les autres propriétés.

Tableau 12–1 Propriétés simples d'aspect de cellule

Propriété	Type	Signification
Spreadsheet	Object	La feuille à laquelle appartient la cellule.
CellBackColor	Long	Couleur du fond.
IsCellBackgroundTransparent	Boolean	True si la couleur de fond n'est pas utilisée.
IsTextWrapped	Boolean	True si le texte est automatiquement renvoyé à la ligne au bord droit de la cellule. False si le texte reste sur une seule ligne (l'affichage peut être tronqué). Valeur par défaut.
ParaIndent	Integer	Retrait du contenu par rapport au bord gauche de la cellule, en 1/100 de mm.
ParaTopMargin	Long	Marge du haut, en 1/100 de mm.
ParaBottomMargin	Long	Marge du bas, en 1/100 de mm.
ParaLeftMargin	Long	Marge de gauche, en 1/100 de mm.
ParaRightMargin	Long	Marge de droite, en 1/100 de mm.

Exemple :

```
rem Code12-02.sxc   bibli : Formater Module5
Option Explicit

Sub FormaterCellule()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim monCurseur As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille2")
maCellule = maFeuille.getCellRangeByName("F16")

maCellule.CellBackColor = RGB(100,220,220)
maCellule.ParaLeftMargin = 200 ' 2 mm de marge gauche
End Sub
```

Format de nombre

Le format d'affichage d'un nombre dans une cellule dépend de la propriété NumberFormat, de type Long. Cette valeur est un index dans la collection des formats disponibles dans le document. Ce concept est décrit plus loin à la section « Les formats de nombre ».

Protection de la cellule

La propriété `CellProtection`, qui n'est active que si la feuille est protégée, est une structure décrite dans le tableau 12-2. Pour modifier un des éléments, vous devez passer par une variable intermédiaire.

Tableau 12–2 Protection de la cellule

Élément	Type	Signification
`IsLocked`	`Boolean`	`True` pour interdire les modifications. Valeur par défaut : `True`.
`IsFormulaHidden`	`Boolean`	`True` pour cacher la formule. Valeur par défaut : `False`.
`IsHidden`	`Boolean`	`True` pour cacher la cellule. Valeur par défaut : `False`.
`IsPrintHidden`	`Boolean`	`True` pour ne pas imprimer la cellule. Valeur par défaut : `False`.

Dans cet exemple, la valeur intermédiaire est initialisée à `False` et seul `IsFormulaHidden` est mis à `True` :

```
Dim laProtection As New com.sun.star.util.CellProtection
laProtection.IsFormulaHidden = true
maCellule.CellProtection = laProtection
```

Alignement horizontal

La propriété `HoriJustify`, de type `Integer`, règle l'alignement horizontal du contenu de la cellule. On lui affecte une des quatre constantes nommées suivantes :

```
com.sun.star.table.CellHoriJustify.STANDARD ' par défaut
com.sun.star.table.CellHoriJustify.LEFT     ' à gauche
com.sun.star.table.CellHoriJustify.CENTER   ' centré
com.sun.star.table.CellHoriJustify.RIGHT    ' à droite
```

La valeur par défaut aligne à gauche pour un nombre, à droite pour un texte.

> Attention à la casse ! Les constantes nommées doivent être écrites en respectant les majuscules et minuscules.

Alignement vertical

La propriété `VertJustify`, de type `Integer`, règle l'alignement vertical du contenu de la cellule. On lui affecte une des quatre constantes nommées suivantes :

```
com.sun.star.table.CellVertJustify.STANDARD ' par défaut
com.sun.star.table.CellVertJustify.TOP      ' en haut
com.sun.star.table.CellVertJustify.CENTER   ' centré
com.sun.star.table.CellVertJustify.BOTTOM   ' en bas
```

L'alignement vertical apparaît si la hauteur de la cellule (ou la ligne) est nettement plus grande que celle des caractères dans la cellule.

Orientation du contenu

Il existe deux manières de définir l'orientation du texte affiché dans la cellule.

La propriété `RotateAngle`, de type `Long`, indique un angle, en 1/100 de degré. Une valeur positive correspond au sens trigonométrique (le sens inverse des aiguilles d'une montre).

La propriété `Orientation`, de type `Integer`, est prise en compte seulement si la propriété `RotateAngle` vaut zéro. Elle reçoit une constante nommée (tableau 12-3) de la forme :

```
com.sun.star.table.CellOrientation.STANDARD
```

Tableau 12–3 Constantes d'orientation du contenu de cellule

Constante	Orientation
STANDARD	Par défaut (de gauche à droite pour le français).
TOPBOTTOM	Pour lire, pencher la tête à droite.
BOTTOMTOP	Pour lire, pencher la tête à gauche.
STACKED	Chaque lettre est horizontale, les lettres sont placées de haut en bas comme une enseigne.

Bordures de la cellule

Il existe une propriété pour chaque bord de la cellule :

- `TopBorder` : bordure du haut ;
- `BottomBorder` : bordure du bas ;
- `LeftBorder` : bordure gauche ;
- `RightBorder` : bordure droite.

Chacune contient une structure `com.sun.star.table.BorderLine` dont les éléments sont listés au tableau 12-4.

Tableau 12–4 Éléments d'une bordure de cellule de tableau

Propriété	Type	Signification
`Color`	`Long`	Couleur de la ligne.
`InnerLineWidth`	`Integer`	Épaisseur de la ligne interne, en 1/100 de mm, dans le cas d'une bordure double. La valeur zéro correspond à une bordure simple.
`OuterLineWidth`	`Integer`	Épaisseur de la ligne simple, ou de la ligne externe dans le cas d'une bordure double ; en 1/100 de mm. La valeur zéro correspond à une bordure inexistante.
`LineDistance`	`Integer`	Distance entre les deux lignes d'une bordure double, en 1/100 de mm.

Un exemple est plus explicite :

```
rem Code12-02.sxc   bibli : Formater Module1
Option Explicit

Sub BorduresDeCellule()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim unBord As New com.sun.star.table.BorderLine
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille2")
maCellule = maFeuille.getCellRangeByName("C2")
With unBord ' utilisez le zoom à 200% !
  .Color = RGB(200,0,0)
  .OuterLineWidth = 30
  maCellule.LeftBorder  = unBord ' ligne simple, rouge
  .OuterLineWidth = 100
  maCellule.RightBorder = unBord ' ligne simple, rouge
  .InnerLineWidth = 60
  .LineDistance = 30
  .Color = RGB(0,120,0)
  maCellule.TopBorder    = unBord ' ligne double, verte
  .Color = RGB(0,0,120)
  maCellule.BottomBorder = unBord ' ligne double, bleue
End With
End Sub
```

Bordures d'un tableau

Pour encadrer une zone rectangulaire de plusieurs cellules et quadriller l'intérieur, inutile de remplir patiemment les bordures de toutes les cellules. On utilise la propriété `TableBorder` d'un objet zone de cellules. Les éléments listés au tableau 12-5 sont très similaires entre eux.

Tableau 12–5 Bordures de tableau

Élément	Type	Signification
`TopLine`	`Object`	Bordure du haut du tableau.
`LeftLine`	`Object`	Bordure de gauche du tableau.
`RightLine`	`Object`	Bordure de droite du tableau.
`BottomLine`	`Object`	Bordure du bas du tableau.
`HorizontalLine`	`Object`	Lignes horizontales du tableau.
`VerticalLine`	`Object`	Lignes verticales du tableau.
`IsTopLineValid`	`Boolean`	`True` si l'élément correspondant est pris en compte.
`IsLeftLineValid`	`Boolean`	`True` si l'élément correspondant est pris en compte.

Tableau 12–5 Bordures de tableau

Élément	Type	Signification
IsRightLineValid	Boolean	True si l'élément correspondant est pris en compte.
IsBottomLineValid	Boolean	True si l'élément correspondant est pris en compte.
IsHorizontalLineValid	Boolean	True si l'élément correspondant est pris en compte.
IsVerticalLineValid	Boolean	True si l'élément correspondant est pris en compte.
Distance	Integer	Non utilisé.
IsDistanceValid	Boolean	Non utilisé.

Voici un exemple qui crée des bordures et un quadrillage de tableau.

```
rem Code12-02.sxc   bibli : Formater Module2
Option Explicit

Sub BorduresDeTableau()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim bordTable As Object
Dim unBord As New com.sun.star.table.BorderLine
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille2")
maZone = maFeuille.getCellRangeByName("B4:D8")
bordTable = maZone.TableBorder
With unBord ' utilisez le zoom à 200% !
  .Color = RGB(200,0,0)
  .OuterLineWidth = 30
  bordTable.LeftLine  = unBord ' ligne simple, rouge
  .OuterLineWidth = 100
  bordTable.RightLine = unBord ' ligne simple, rouge
  .OuterLineWidth = 30
  .Color = RGB(220,220,0) ' quadrillage de lignes jaunes
  bordTable.VerticalLine   = unBord
  bordTable.HorizontalLine = unBord
  .InnerLineWidth = 60
  .LineDistance = 30
  .Color = RGB(0,120,0)
  bordTable.TopLine    = unBord ' ligne double, verte
  .Color = RGB(0,0,120)
  bordTable.BottomLine = unBord ' ligne double, bleue
End With
With bordTable
  .IsBottomLineValid = true
  .IsTopLineValid = true
  .IsLeftLineValid = true
  .IsRightLineValid = true
  .IsHorizontalLineValid = true
```

```
  .IsVerticalLineValid = true
End With
maZone.TableBorder = bordTable
End Sub
```

Les propriétés `IsHorizontalLineValid` et `IsVerticalLineValid` ont une signification particulière, car elles concernent le quadrillage intérieur.

- En lecture, la valeur `True` indique que toutes les lignes de ce type ont les mêmes valeurs de bordure.
- En écriture, la valeur `True` indique que toutes les lignes de ce type doivent prendre la même valeur de bordure.

Ombre de la cellule

La propriété `ShadowFormat` est une structure `com.sun.star.table.ShadowFormat` comportant plusieurs éléments, voir le tableau 12-6.

Tableau 12–6 Structure de ShadowFormat

Propriété	Type	Signification
`Location`	`Integer`	Position de l'ombre, sous forme de constante nommée, voir tableau suivant.
`ShadowWidth`	`Integer`	Largeur de l'ombre, en 1/100 de mm.
`IsTransparent`	`Boolean`	`True` si l'ombre est transparente.
`Color`	`Long`	Couleur de l'ombre.

La position de l'ombre est exprimée sous forme de constante nommée (tableau 12-7), de la forme :

```
com.sun.star.table.ShadowLocation.BOTTOM_RIGHT
```

Tableau 12–7 Constantes de position d'ombre de cellule

Constante	Signification
`NONE`	Aucune ombre.
`TOP_LEFT`	Ombre portée vers le haut et à gauche.
`TOP_RIGHT`	Ombre portée vers le haut et à droite.
`BOTTOM_LEFT`	Ombre portée vers le bas et à gauche.
`BOTTOM_RIGHT`	Ombre portée vers le bas et à droite.

Il est nécessaire de remplir la structure dans une variable de travail qui servira à initialiser `ShadowFormat` comme ici :

```
rem Code12-02.sxc   bibli : Formater Module3
Option Explicit

Sub OmbreDeCellule()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim ombre As New com.sun.star.table.ShadowFormat
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille2")
maCellule = maFeuille.getCellRangeByName("E2")
ombre.Location = com.sun.star.table.ShadowLocation.TOP_LEFT
ombre.ShadowWidth = 200 ' environ 3,5 mm
ombre.Color = RGB(100,100,100) ' couleur grise
maCellule.ShadowFormat = ombre
End Sub
```

Pour bien voir l'ombre, il est préférable que la cellule soit encadrée ou ressorte avec une couleur de fond.

Note (annotation) de cellule

Si une cellule comporte une note, celle-ci est obtenue avec la propriété `Annotation`, qui contient elle-même plusieurs éléments listés dans le tableau 12-8.

Tableau 12–8 Éléments d'une note de cellule

Élément	Type	Signification
`String`	`String`	Texte de la note.
`Author`	`String`	Auteur de la note. Lecture seulement.
`Date`	`String`	Date de création de la note. Lecture seulement.
`IsVisible`	`Boolean`	`True` si la note est affichée en permanence. `False` si la note ne s'affiche qu'au passage de la souris.
`Position`	`Object`	Position de la cellule qui contient cette note. Lecture seulement.

Si la cellule ne comporte pas de note, les propriétés `String`, `Author`, `Date` contiennent des chaînes nulles. La macro ci-dessous illustre ces propriétés.

```
rem Code12-02.sxc   bibli : Annoter Module1
Option Explicit

Sub LireNote()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim laNote As Object, texteNote As String, cr As String
cr = chr(13) ' fin de ligne
monDocument = thisComponent
```

```
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille3")
maCellule = maFeuille.getCellRangeByName("B3")

laNote = maCellule.Annotation
MsgBox("Contenu de la note : " & laNote.String & cr & _
       "Auteur de la note : "  & laNote.Author & cr & _
       "Date de la note : "    & laNote.Date & cr & _
       "Visible en permanence : " & laNote.IsVisible)
if MsgBox("Changer la note ?", 4) = 6 then
  laNote.String = "Nouveau" & cr & "texte" & cr & _
                  "avec" & cr & "quatre lignes"
end if
End Sub
```

Il est possible de définir un curseur d'écriture pour écrire le texte d'une note, mais, en l'absence de possibilités de formatage et compte tenu de l'habituelle simplicité des notes, il est plus simple de manipuler directement la propriété `String`.

Les notes qui existent dans une feuille se trouvent dans la collection `Annotations` (attention au « s ») de la feuille. La macro suivante va énumérer les notes présentes. Elle utilise la routine utilitaire `adresseString`, décrite à l'annexe B, qui convertit une adresse de cellule en adresse sous forme de texte.

```
rem Code12-02.sxc   bibli : Annoter Module2
Option Explicit

Sub ListerNotes()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim laNote As Object, texteNote As String, cr As String
Dim lesNotes As Object, x As Long
cr = chr(13) ' fin de ligne
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille3")

lesNotes = maFeuille.Annotations
for x = 0 to lesNotes.Count -1
  laNote = lesNotes(x)
  MsgBox("Contenu de la note : " & laNote.String & cr & _
     "Auteur de la note : "  & laNote.Author & cr & _
     "Date de la note : "    & laNote.Date & cr & _
     "Visible en permanence : " & laNote.IsVisible & cr & _
     "Adresse de la cellule : " & adresseString(monDocument, laNote.Position))

next
End Sub
```

Lignes et colonnes

Une zone de cellules délimite un ensemble de lignes et un ensemble de colonnes qui traversent la zone, cette dernière pouvant se réduire à une seule cellule. Les possibilités sont identiques pour les lignes et les colonnes, comme nous allons le voir.

Les lignes

À partir d'une zone de cellules, on obtient une collection de lignes grâce à la propriété `Rows` de l'objet zone. Cette collection de lignes est un objet. On accède à chacune des lignes en indexant l'objet collection de lignes ; l'index zéro correspond à la première ligne de la zone.

```
Dim lesLignes As Object, uneLigne As Object

maZone = uneFeuille.getCellRangeByName("E3:J7")
lesLignes = maZone.Rows
print "Nombre de lignes : " & lesLignes.Count
uneLigne = lesLignes(1) ' deuxième ligne de la zone
```

Avec un objet ligne, il est possible d'effectuer des changements, par exemple de formatage, sur l'ensemble des cellules concernées, comme avec un objet zone de cellules.

Les propriétés listées dans le tableau 12-9 sont disponibles aussi bien pour une ligne que pour une collection de lignes. Elles peuvent être lues et modifiées.

Tableau 12–9 Propriétés de lignes

Propriété	Type	Signification
`IsVisible`	`Boolean`	`True` si la ligne est visible, `False` si elle est cachée.
`Height`	`Long`	Hauteur de la ligne en 1/100 de mm.
`OptimalHeight`	`Boolean`	`True` : la ligne adapte sa hauteur à son contenu. `False` : la hauteur dépend de la propriété `Height`.

La méthode `insertByIndex` de l'objet collection de lignes ajoute plusieurs lignes. Dans cet exemple, on ajoute trois lignes adjacentes dont la première aura le rang 1 dans la collection :

```
lesLignes.insertByIndex(1,3)
```

La méthode `removeByIndex` de l'objet collection de lignes supprime plusieurs lignes. Dans cet exemple, on supprime trois lignes adjacentes dont la première aura le rang 1 dans la collection :

```
lesLignes.removeByIndex(1,3)
```

Les colonnes

À partir d'une zone de cellules, on obtient une collection de colonnes grâce à la propriété `Columns` de l'objet zone. Cette collection de colonnes est un objet. On accède à chacune des colonnes en indexant l'objet collection de colonnes ; l'index zéro correspond à la première colonne de la zone.

```
Dim lesCols As Object, uneCol As Object

maZone = uneFeuille.getCellRangeByName("E3:J7")
lesCols = maZone.Columns
print "Nombre de colonnes : " & lesCols.Count
uneCol = lesCols(1) ' deuxième colonne de la zone
```

Avec un objet colonne, il est possible d'effectuer des changements, par exemple de formatage, sur l'ensemble des cellules concernées, comme avec un objet zone de cellules.

Les propriétés listées dans le tableau 12-10 sont disponibles aussi bien pour une colonne que pour une collection de colonnes. Elles peuvent être lues et modifiées.

Tableau 12–10 Propriétés de colonnes

Propriété	Type	Signification
`IsVisible`	`Boolean`	`True` si la colonne est visible, `False` si elle est cachée.
`Width`	`Long`	Largeur de la colonne en 1/100 de mm.
`OptimalWidth`	`Boolean`	`True` : la colonne adapte sa largeur à son contenu. `False` : la largeur dépend de la propriété `Width`.

La méthode `insertByIndex` de l'objet collection de colonnes ajoute plusieurs colonnes. Dans cet exemple, on ajoute trois colonnes adjacentes dont la première aura le rang 1 dans la collection :

```
lesCols.insertByIndex(1,3)
```

La méthode `removeByIndex` de l'objet collection de colonnes supprime plusieurs colonnes. Dans cet exemple, on supprime trois colonnes adjacentes dont la première aura le rang 1 dans la collection :

```
lesCols.removeByIndex(1,3)
```

Lire et écrire dans une cellule

Les différents contenus d'une cellule

Le contenu d'une cellule se présente sous une des quatre formes suivantes :

- cellule vide ;
- valeur numérique (un nombre) ;
- un texte (qui peut être très long) ;
- une formule (qui s'évalue comme une valeur numérique ou un texte).

La propriété `Type`, de type `Long`, fournit le type de contenu de la cellule sous forme d'une constante nommée de la forme :

```
com.sun.star.table.CellContentType.TEXT
```

Vous trouverez la liste complète de ces constantes dans le prochain exemple.

La propriété `String` de la cellule fournit le texte affiché par celle-ci, sous forme d'une chaîne de caractères (autrement dit, la propriété `String` est du type `String` !). Pour une valeur numérique, la propriété `String` tient compte du format du nombre, par exemple le nombre de décimales.

Quand la cellule contient un nombre ou une formule qui s'évalue en nombre, sa propriété `Value` renvoie la valeur ; cette propriété est du type `Double`, même si visuellement la cellule affiche un nombre entier ou une date.

Quand la cellule contient une formule (c'est-à-dire que son contenu débute par le signe `=`), la propriété `Formula`, de type `String`, renvoie l'expression de la formule en langue anglaise. La propriété `FormulaLocal`, elle, renvoie l'expression sous sa forme localisée. Si la formule renvoie un texte, la propriété `Value` contient zéro, même si le texte obtenu est celui d'un nombre. Si la formule renvoie une valeur numérique, la propriété `String` contient un texte représentant la valeur.

Pour bien comprendre ces propriétés, le mieux est de les afficher pour divers contenus de cellule. Cliquez sur une cellule de tableur et exécutez la macro ci-après. Le fichier du zip téléchargeable contenant la macro comporte des cellules intéressantes sur la feuille `Contenus`.

```
rem Code12-01.sxc   bibli : UneCellule Module1
Option Explicit

Sub ContenuDeCellule() ' pointez une cellule sur une feuille
Dim monDocument As Object, maCellule As Object
Dim mess As String, cr As String
```

```
cr = chr(13) ' retour à la ligne
monDocument = thisComponent
maCellule = monDocument.currentSelection
if maCellule.supportsService("com.sun.star.table.Cell") then
  Select Case maCellule.Type
  Case com.sun.star.table.CellContentType.EMPTY
    mess = "La cellule est vide"
  Case com.sun.star.table.CellContentType.TEXT
    mess = "La cellule contient du texte"
  Case com.sun.star.table.CellContentType.VALUE
    mess = "La cellule contient une valeur"
  Case com.sun.star.table.CellContentType.FORMULA
    mess = "La cellule contient une formule"
    if maCellule.Value = 0  then
      if maCellule.String <> "0"  then
        mess = "La cellule contient une formule donnant un texte"
      else
        mess = "La cellule contient une formule donnant zéro"
      end if
    else
      mess = "La cellule contient une formule donnant une valeur"
    end if
  End Select
  MsgBox mess & cr & "String : " & maCellule.String & cr & _
    "Value : " & maCellule.Value & cr & _
    "Formula : " &  maCellule.Formula & cr & _
    "FormulaLocal : " &  maCellule.FormulaLocal & cr
else
  MsgBox("Une seule cellule SVP !", 16)
end if
End Sub
```

Les propriétés de cellule qui permettent de connaître le contenu sont aussi utilisables pour remplir une cellule. Une seule de ces instructions suffit, la propriété `Type` sera mise à jour automatiquement :

```
maCellule.String = "Un petit texte"
maCellule.Value = 12345
maCellule.Formula = "=A3+B17"
maCellule.FormulaLocal= "=Arrondi(F37/B12)"
```

Le curseur d'écriture dans la cellule

Si vous savez comment écrire du texte par macro dans un document Writer, vous retrouverez ici les mêmes concepts, mais pas toutes les possibilités.

Nous venons de voir la méthode la plus simple pour lire ou écrire un texte dans une cellule : utiliser sa propriété `String`. Pour avoir plus de possibilités, il faut utiliser un cur-

seur d'écriture, qui pointe sur un endroit dans le texte de la cellule : le point d'insertion. On obtient un curseur d'écriture à partir de l'objet cellule :

```
Dim monCurseur As Object
monCurseur = maCellule.createTextCursor
```

À sa création, le curseur d'écriture pointe *à la fin du texte* contenu dans la cellule.

Déplacer le curseur d'écriture

L'objet curseur dispose de plusieurs méthodes permettant de le déplacer. Certaines renvoient un résultat :

- `True` si l'action a pu être réalisée,
- `False` dans le cas contraire.

En pratique, on utilise rarement le résultat de ces fonctions et on les utilise comme des méthodes de type `Sub`.

Les fonctions de déplacement ont toutes un argument booléen, que nous désignerons par `SEL`, qui a l'effet suivant :

- `SEL = False` : le curseur se déplace (comme la barre verticale du curseur visible quand vous éditez le texte d'une cellule).
- `SEL = True` : le curseur se déplace en étendant la sélection (c'est le même effet qu'une sélection progressive du curseur visible en faisant glisser la souris).

Et voici un petit exemple :

```
Dim monDocument As Object
Dim lesFeuilles As Object, maFeuille As Object
Dim maCellule As Object, monCurseur As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Ecrire")
maCellule = maFeuille.getCellRangeByName("C2")
monCurseur = maCellule.createTextCursor
' ici le curseur est à la fin du texte
monCurseur.gotoStart(False)
' ici le curseur est au début du texte
```

Le tableau 12-11 liste les fonctions de déplacement de curseur.

Tableau 12–11 Déplacement du curseur d'écriture

Méthode	Effet sur le curseur
`goRight(n,SEL)`	Déplacer de n caractères à droite. Renvoie `True` si l'action a été réalisée.
`goLeft(n,SEL)`	Déplacer de n caractères à gauche. Renvoie `True` si l'action a été réalisée.
`gotoStart(SEL)`	Déplacer au début du texte de la cellule.
`gotoEnd(SEL)`	Déplacer à la fin du texte de la cellule.

Nous avons signalé que le curseur d'écriture peut sélectionner une zone. Après avoir effectué une action sur cette zone, on dispose de deux méthodes du curseur pour le ramener à un point d'insertion, situé au début ou bien à la fin de la zone.

```
monCurseur.collapseToStart ' début de zone
monCurseur.collapseToEnd   ' fin de zone
```

L'objet curseur de cellule fournit aussi une fonction booléenne `isCollapsed`, qui renvoie `False` si le curseur est étendu pour une sélection et `True` s'il est ponctuel.

```
if monCurseur.isCollapsed then
  ' ici le curseur est ponctuel
end if
```

Créer un curseur à partir d'un autre curseur

Dans certains cas, il est intéressant d'utiliser un autre curseur d'écriture qui soit initialisé aux valeurs du curseur actuel. On utilise pour cela la méthode `createTextCursorByRange` de l'objet `cellule`. Dans cet exemple, `curseur2` reprend la position et la sélection dans le texte mémorisées par `curseur1` :

```
Dim monDocument As Object
Dim lesFeuilles As Object, maFeuille As Object
Dim maCellule As Object
Dim curseur1 As Object, curseur2 As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Ecrire")
maCellule = maFeuille.getCellRangeByName("C2")
curseur1 = maCellule.createTextCursor
curseur2 = maCellule.createTextCursorByRange(curseur1)
' curseur2 est à la fin du texte
curseur1.gotoStart(False)
' curseur1 est au début du texte
```

Les objets `curseur1` et `curseur2` peuvent être déplacés ou modifiés indépendamment l'un de l'autre.

Si le premier curseur sélectionne une zone, le deuxième curseur peut être créé à partir du premier en précisant qu'il doit se positionner à la fin de la zone de texte sélectionnée :

```
curseur2 = maCellule.createTextCursorByRange(curseur1.End)
```

Pour le positionner au début de la zone sélectionnée, on écrirait :

```
curseur2 = maCellule.createTextCursorByRange(curseur1.Start)
```

En fait, il est possible de définir un curseur à partir d'un objet zone de texte (`TextRange`). Cet objet désigne une partie de texte, mais il ne possède pas les méthodes d'un objet curseur.

Lire un texte dans une cellule

Nous savons récupérer le texte entier contenu dans une cellule. En utilisant le déplacement du curseur d'écriture, nous allons récupérer une partie du texte.

```
rem Code12-01.sxc   bibli : UneCellule Module2
Option Explicit

Sub LireUnMorceau()
Dim monDocument As Object
Dim lesFeuilles As Object, maFeuille As Object
Dim maCellule As Object, monCurseur As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Ecrire")
maCellule = maFeuille.getCellRangeByName("C2")
monCurseur = maCellule.createTextCursor
monCurseur.goLeft(4, true)
print monCurseur.String
End Sub
```

La macro crée un curseur dans une cellule ; il se situe alors en fin de texte ; le déplacement à gauche sélectionne les quatre caractères précédents. La macro affiche ensuite le contenu de la propriété `String` (de type `String`) de l'objet curseur (ne pas confondre avec la propriété `String` de l'objet cellule).

Insérer un texte dans une cellule

Ici, nous allons insérer un texte dans un texte existant. La première méthode consiste à utiliser la propriété `String` du curseur. En reprenant l'exemple précédent, ajouter la ligne suivante remplacera le texte sélectionné par un nouveau texte :

```
monCurseur.String = "Terre"
```

Si le curseur était ponctuel, le texte serait inséré au point d'insertion. Cependant, pour ajouter un autre texte à la suite, vous devez déplacer le curseur. La deuxième méthode effectue un déplacement automatique du curseur.

```
maCellule.insertString(monCurseur,  "blabla", SEL)
```

La méthode `insertString` de l'objet cellule utilise trois arguments. Le premier est un objet curseur d'écriture ; le deuxième est la chaîne de caractères à insérer.

Le troisième argument reçoit en général la valeur `False`. La valeur `True` est employée pour remplacer une zone préalablement sélectionnée avec le curseur. Le curseur redevient ponctuel et se positionne à droite du texte ajouté. Un nouvel appel de la méthode ajoutera du texte dans le sens d'écriture. L'exemple suivant suppose une cellule contenant le texte « Un navire », que la macro va transformer en « Il était un petit navire ».

```
rem Code12-01.sxc   bibli : UneCellule Module3
Option Explicit

Sub InsererTexte()
Dim monDocument As Object , tx1 As String
Dim lesFeuilles As Object, maFeuille As Object
Dim maCellule As Object, monCurseur As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Ecrire")
maCellule = maFeuille.getCellRangeByName("C5")
monCurseur = maCellule.createTextCursor
monCurseur.gotoStart(false)
monCurseur.goRight(2, true) ' sélectionner "Un"
tx1 = "Il était un"
maCellule.insertString(monCurseur, tx1, true)
maCellule.insertString(monCurseur, " petit", false)
End Sub
```

Insérer des caractères spéciaux

La méthode `insertControlCharacter` de l'objet cellule permet d'insérer une marque de paragraphe. L'exemple suivant utilise une cellule contenant déjà un texte et ajoute un nouveau paragraphe suivi d'un texte.

```
rem Code12-01.sxc   bibli : UneCellule Module4
Option Explicit

Sub InsererParagraphe()
Dim monDocument As Object
Dim lesFeuilles As Object, maFeuille As Object
Dim maCellule As Object, monCurseur As Object
Dim special As Integer
special = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Ecrire")
maCellule = maFeuille.getCellRangeByName("C2")
monCurseur = maCellule.createTextCursor
maCellule.insertControlCharacter(monCurseur, special, false)
maCellule.insertString(monCurseur, "Mon ami Pierrot", false)
End Sub
```

Le premier argument de `insertControlCharacter` est un objet curseur.

Le deuxième argument est une valeur numérique du type `Integer`, qui se définit par une constante nommée. Deux valeurs seulement sont utilisables.

```
com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
com.sun.star.text.ControlCharacter.APPEND_PARAGRAPH
```

Avec la première valeur, la marque de paragraphe est insérée à la position du curseur. Avec la deuxième valeur, la marque de paragraphe est insérée à la fin du paragraphe dans lequel se trouve le curseur, puis le curseur est positionné au début de ce nouveau paragraphe.

Formater la cellule

Imposer son style

Le plus rapide pour formater une cellule est d'utiliser un style de cellule déjà défini dans le document. La propriété `CellStyle` est une chaîne de caractères qui contient le nom du style. On peut facilement changer le nom du style en cours :

```
print maCellule.CellStyle
maCellule.CellStyle= "MonStyleAMoi"
```

> Attention à la casse ! Les noms de styles doivent être écrits en respectant les majuscules et minuscules.

On peut modifier de la même manière le style d'une zone de cellules.

Inversement, la propriété `CellStyle` fournit le nom du style de la cellule. Pour une zone comportant plusieurs styles, la propriété `CellStyle` fournira une chaîne de caractères nulle.

> **PIÈGE Les noms de styles traduits**
>
> Pour les styles standards fournis avec OpenOffice.org, vous récupérez dans `CellStyle` le nom anglais du style, même avec une version francisée. Par exemple, en affectant le style « Standard » on relira le style « Default ». En revanche, les styles que vous créez n'ont évidemment qu'un seul nom. L'annexe B offre une fonction `getLocaleStyleName`, qui traduit un nom de style anglais dans son nom localisé.

Formatage local de la cellule

Les possibilités de formatage qui suivent sont spécifiques à la cellule concernée, elles ne modifient pas le style de la cellule. De plus, une interrogation du style de la cellule donnera toujours le même nom, alors que la cellule est en réalité formatée différemment.

L'objet cellule accepte la plupart des propriétés de formatage de caractère disponibles dans un document Writer. Lorsque vous imposez un formatage avec la cellule, il s'appliquera à l'ensemble des caractères affichés dans la cellule. Cependant, avec le curseur d'écriture dans la cellule, vous pouvez aussi formater certains caractères : il suffit de les sélectionner avec le curseur, puis d'appliquer la propriété de formatage avec le curseur.

Nous nous contenterons de lister les propriétés principales dans le tableau 12-12 et de fournir un exemple récapitulatif. Vous trouverez le détail de ces propriétés au chapitre 11, section « Formatage local des caractères ».

Tableau 12–12 Propriétés de formatage de caractère

Propriété	Type	Signification
`CharWeight`	`Integer`	La « graisse » du caractère.
`CharPosture`	`Integer`	Italique.
`CharUnderline`	`Integer`	Soulignement.
`CharUnderlineColor`	`Long`	Couleur du soulignement. La valeur -1 correspond à la couleur « Automatique ».
`CharUnderlineHasColor`	`Boolean`	`True` si le soulignement possède sa propre couleur. `False` si le soulignement a la même couleur que le caractère.
`CharColor`	`Long`	Couleur de la police de caractère. La valeur -1 correspond à la couleur « Automatique ».
`CharFontName`	`String`	Nom de la police de caractère (respecter la casse).
`CharHeight`	`Integer`	Taille du caractère, en points.
`CharShadowed`	`Boolean`	Valeur `True` pour ombrer le caractère.
`CharCrossedOut`	`Boolean`	Valeur `True` pour barrer le caractère.
`CharLocale`	`Object`	Région utilisée pour les fonctions dépendant de la localisation.

Un exemple :

```
rem Code12-02.sxc   bibli : Formater Module4
Option Explicit

Sub FormaterCaractere()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim monCurseur As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille2")
maCellule = maFeuille.getCellRangeByName("B13")
```

```
With maCellule
  .CharHeight = 14
  .CharColor = RGB(0,100,100) ' couleur bleu-vert
  .CharUnderline =  com.sun.star.awt.FontUnderline.DASHDOT
End With
maCellule = maFeuille.getCellRangeByName("D15")
monCurseur = maCellule.createTextCursor ' positionné à la fin
monCurseur.goLeft(3, true) ' sélectionner les 3 derniers car.
monCurseur.CharColor = RGB(200,0,0) ' les peindre en rouge
End Sub
```

L'objet zone de cellules accepte les mêmes propriétés de formatage que l'objet cellule. La même instruction de formatage s'applique alors à toutes les cellules de la zone.

Les formats de nombre

Différents formats de nombre sont disponibles par défaut dans chaque document Calc. L'utilisateur peut en ajouter d'autres, qui ne seront connus que dans ce document. La collection des formats disponibles est gérée par l'objet `NumberFormats`, contenu dans l'objet document. Chaque format est accessible par une clé, qui est une simple valeur numérique (un index). Le tableau 12-13 liste les propriétés d'un format.

Tableau 12–13 Propriétés d'un format de nombre

Propriété	Type	Signification
`FormatString`	`String`	Le format exprimé sous forme d'une chaîne de caractères.
`Locale`	`Object`	Langue et et variante de pays.
`Type`	`Integer`	Type de format de nombre; somme de constantes nommées, voir explications .
`Comment`	`String`	Commentaire sur le format, pour l'utilisateur.

La propriété `Locale` est une structure `com.sun.star.lang.Locale` comportant essentiellement les éléments :

- `Language` : langue employée, suivant le code ISO 639 à deux lettres minuscules ; une liste en français se trouve à l'adresse http://www.termisti.refer.org/iso639.htm.
- `Country` : variante de pays, suivant le code ISO 3166 à deux lettres majuscules ; une liste en français se trouve à l'adresse http://whois.perl-gratuit.com/documentation/codes_iso.html.

La valeur `Par défaut` correspond à des chaînes nulles pour `Language` et `Country`.

L'ensemble des clés existantes est renvoyé par la méthode `queryKeys` de l'objet collection des formats et la méthode `getByKey` renvoie le format correspondant à une clé.

La valeur de `Type` d'un format est la somme de constantes nommées, qui sont des puissances de deux. Elles sont listées dans la macro `TypeFormat` ci-après. La valeur `ALL` est un cas particulier : sa valeur est zéro. La valeur `DATETIME` est égale à `DATE+TIME`.

La macro `ListerFormats` utilise ces facilités pour afficher dans une feuille de tableur la liste des formats, pour une valeur de `Locale`.

```
rem Code12-05.sxc   bibli : FormatNombres Module1
Option Explicit

Sub ListerFormats()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object, maZone As Object
Dim lesFormats As Object, unFormat As Object
Dim lesCles As Variant, x As Long
Dim patois As New com.sun.star.lang.Locale
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Listage")
maZone = maFeuille.getCellRangeByName("A2:F100")
maZone.clearContents(com.sun.star.sheet.CellFlags.STRING + _
                     com.sun.star.sheet.CellFlags.VALUE )

patois.Language = InputBox("Langue (rien, fr, es, de...)")
patois.Country = InputBox("Variante (rien, FR, ES, DE, LU...)")

lesFormats = monDocument.NumberFormats
lesCles = lesFormats.queryKeys(_
          com.sun.star.util.NumberFormat.ALL, patois, false)
for x = LBound(lesCles) to UBound(lesCles)
  unFormat = lesFormats.getByKey(lesCles(x))
  maCellule = maFeuille.getCellByPosition(0,1+x)
  maCellule.Value = lesCles(x) ' numéro de la clé
  maCellule = maFeuille.getCellByPosition(1,1+x)
  maCellule.String = unFormat.FormatString
  maCellule = maFeuille.getCellByPosition(2,1+x)
  maCellule.String = unFormat.Locale.Language
  maCellule = maFeuille.getCellByPosition(3,1+x)
  maCellule.String = unFormat.Locale.Country
  maCellule = maFeuille.getCellByPosition(4,1+x)
  maCellule.String = TypeFormat(unFormat.Type)
  maCellule = maFeuille.getCellByPosition(5,1+x)
  maCellule.String = unFormat.Comment
next
End Sub
```

```
Function TypeFormat(leType As Integer) As String
Dim resu As String
if leType = com.sun.star.util.NumberFormat.ALL then
  resu = "ALL"
else
  if (leType AND com.sun.star.util.NumberFormat.DEFINED) then
    resu = resu & "DEFINED "
  end if
  if (leType AND com.sun.star.util.NumberFormat.DATE) then
    resu = resu & "DATE "
  end if
  if (leType AND com.sun.star.util.NumberFormat.TIME) then
    resu = resu & "TIME "
  end if
  if (leType AND com.sun.star.util.NumberFormat.CURRENCY) then
    resu = resu & "CURRENCY "
  end if
  if (leType AND com.sun.star.util.NumberFormat.NUMBER) then
    resu = resu & "NUMBER "
  end if
  if (leType AND com.sun.star.util.NumberFormat.SCIENTIFIC) then
    resu = resu & "SCIENTIFIC "
  end if
  if (leType AND com.sun.star.util.NumberFormat.FRACTION) then
    resu = resu & "FRACTION "
  end if
  if (leType AND com.sun.star.util.NumberFormat.PERCENT) then
    resu = resu & "PERCENT "
  end if
  if (leType AND com.sun.star.util.NumberFormat.TEXT) then
    resu = resu & "TEXT "
  end if
  if (leType AND com.sun.star.util.NumberFormat.DATETIME) then
    resu = resu & "DATETIME "
  end if
  if (leType AND com.sun.star.util.NumberFormat.LOGICAL) then
    resu = resu & "LOGICAL "
  end if
  if (leType AND com.sun.star.util.NumberFormat.UNDEFINED) then
    resu = resu & "UNDEFINED "
  end if
end if
TypeFormat = resu
End Function
```

La macro principale commence par effacer la zone qui va recevoir la liste des formats, en utilisant la méthode `clearContents` qui est décrite plus loin, section « Effacer une zone de cellules ». La langue et sa variante de pays sont choisies par l'utilisateur.

La collection de clés est récupérée sous forme d'un tableau de valeurs, que nous explorons avec la boucle `next`. Pour chacune, les différents éléments sont mémorisés dans les cellules de colonne A à F, une ligne par clé.

Le type de format étant codé par une constante nommée ou l'addition de plusieurs de ces constantes, la fonction `TypeFormat` traduit la valeur numérique dans une chaîne de caractères représentant ce ou ces type(s). Pour cela, on teste l'existence de chacune des constantes nommées possibles, qui sont des valeurs binaires avec un seul bit à 1.

Exécutez la macro `ListerFormats` avec diverses valeurs du `Locale`. Essayez en particulier les valeurs : `rien, rien ; fr, rien ; fr, FR ; fr, LU ; en, GB ; en, US ; es ; de`. Vous constaterez que :

- Les valeurs de clés dépendent du `Locale` et il y a des trous dans la numérotation.
- Le nombre de clés disponibles dépend du `Locale`.
- La chaîne de caractères `FormatString` dépend aussi du `Locale`.
- Les formats définis par l'utilisateur sont ajoutés en fin de liste pour un `Locale` donné (nous avons défini un format pour `Locale rien, rien` et un format pour `Locale fr, LU`).

Conclusion : il est risqué d'affecter à une cellule une clé de format si on ne peut être sûr du document sur lequel on travaille.

ATTENTION

Même une clé de format disponible par défaut peut dépendre de la version OpenOffice.org avec laquelle le document a été créé.

On vérifie l'existence d'une chaîne de format avec la méthode `queryKey` (notez le singulier) :

```
rem Code12-05.sxc   bibli : FormatNombres Module2
Option Explicit

Sub ChaineFormatVersCle()
Dim monDocument As Object
Dim laCle As Long, monFormat As String, lesFormats As Object
Dim patois As New com.sun.star.lang.Locale
monDocument = thisComponent

patois.Language = InputBox("Langue (rien, fr, es, de...)")
patois.Country = InputBox("Variante (rien, FR, ES, DE, LU...)")
monFormat = InputBox("Chaîne de format recherchée")

lesFormats = monDocument.NumberFormats
laCle = lesFormats.queryKey(monFormat, patois, false)
if laCle < 0 then
  print "Pas de clé pour ce format"
```

```
else
  print "Clé : " & laCle
end if
End Sub
```

Le troisième argument de `queryKey` n'est pas utilisé actuellement ; il doit être mis à `False`. Si vous n'avez pas retrouvé votre format préféré, vous pouvez l'ajouter dans la liste.

```
rem Code12-05.sxc   bibli : FormatNombres Module3
Option Explicit

Sub AjouterFormat()
Dim monDocument As Object
Dim laCle As Long, monFormat As String, lesFormats As Object
Dim patois As New com.sun.star.lang.Locale
monDocument = thisComponent

patois.Language = InputBox("Langue (rien, fr, es, de...)")
patois.Country = InputBox("Variante (rien, FR, ES, DE, LU...)")
monFormat = InputBox("Chaîne de format à ajouter")
lesFormats = monDocument.NumberFormats
' ajouter un format qui n'existe pas déjà
laCle = lesFormats.addNew(monFormat, patois)
print "Nouvelle clé : " & laCle
print "Ce format va être supprimé"
lesFormats.removeByKey(laCle)
End Sub
```

Une erreur est déclenchée si vous tentez d'ajouter un format existant pour le `Locale`.

Une fois en possession de votre clé, formater une cellule ou une zone est un jeu d'enfant :

```
maCellule.NumberFormat = laCle
maZone.NumberFormat = laCle
```

Les mécanismes de format numérique sont très riches et complexes. Pour aller plus loin, consultez le chapitre 6.2.5 du « Developer's Guide » du SDK, fourni dans le zip téléchargeable.

Méthodes pratiques

Dans cette partie, nous allons aborder diverses possibilités des macros dans un document Calc.

Activer le calcul des formules

Le calcul des valeurs de cellules est normalement effectué automatiquement après la modification d'une cellule et pour les cellules qui dépendent de celle-là. Dans un document tableur assez complexe, le temps de recalcul des cellules peut être non négligeable et ralentir l'exécution des macros. L'objet document nous donne plusieurs moyens de maîtriser les conditions de recalculs.

Déterminer si le recalcul automatique est actif :

```
if monDocument.isAutomaticCalculationEnabled then
   ' le recalcul automatique est actuellement activé
end if
```

Inhiber, puis activer le recalcul automatique :

```
'  inhiber le recalcul automatique
monDocument.enableAutomaticCalculation(false)
'    - - effectuer divers travaux - -
'  activer le recalcul automatique
monDocument.enableAutomaticCalculation(true)
```

Recalculer seulement les formules qui ne sont pas à jour :

```
monDocument.calculate
```

Recalculer toutes les formules du tableur

```
monDocument.calculateAll
```

Effacer une zone de cellules

La méthode `clearContents` d'un objet zone de cellules est très pratique pour réinitialiser toute une zone selon certains critères, qui sont spécifiés avec des constantes nommées de la forme :

```
com.sun.star.sheet.CellFlags.STRING
```

Ces constantes sont listées au tableau 12-14. On peut combiner plusieurs critères en les additionnant : le résultat est équivalent à utiliser `clearContents` avec la première constante, puis la deuxième, etc.

Tableau 12–14 Critères de choix des cellules

Constante	Signification
VALUE	Cellules ayant un contenu numérique, sauf les contenus au format de date ou heure.
DATETIME	Cellules ayant un contenu numérique au format de date ou heure.
STRING	Cellules ayant un contenu de texte.
ANNOTATION	Cellules contenant une note.
FORMULA	Cellules contenant une formule (pas une simple valeur numérique).
HARDATTR	Cellules ayant un formatage spécifique (autre que celui du style de la cellule).
STYLES	(fonctionnement actuellement identique à HARDATTR)
OBJECTS	Cellules contenant un objet de dessin (le dessin doit être entièrement dans la zone analysée).
EDITATTR	Cellules comportant un formatage sur une partie du texte.

À titre d'exemple, nous allons effacer le formatage local des cellules et supprimer leur texte.

```
rem Code12-03.sxc   bibli : Fonctions Module5
Option Explicit

Sub EffacerZone()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim gomme As Long
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Fonctions")
maZone = maFeuille.getCellRangeByName("B30:D33")

gomme = com.sun.star.sheet.CellFlags.HARDATTR + _
        com.sun.star.sheet.CellFlags.STRING
maZone.clearContents(gomme)
End Sub
```

Énumérer les cellules d'une zone

La méthode que nous allons exposer est applicable à toute sélection, y compris des cellules sélectionnées par l'utilisateur. Cet exemple, typique des méthodes d'énumération, recherche dans la zone toutes les cellules contenant du texte et met ce texte en majuscules.

```
rem Code12-01.sxc   bibli : ZonesCellules Module6
Option Explicit
```

```
Sub CellulesMajuscules()
Dim controleur As Object, maFeuille As Object
Dim sel As Object, listeCell As Object, selectionCell As Object
Dim uneCellule As Object, enumCellules As Object

controleur = ThisComponent.CurrentController
maFeuille = controleur.ActiveSheet
sel = ThisComponent.CurrentSelection
'sel = maFeuille.getCellRangeByName("NomDeZone")
'sel = maFeuille.getCellRangeByName("C11:D12")
selectionCell = sel.queryContentCells(com.sun.star.sheet.CellFlags.STRING)
enumCellules = selectionCell.getCells
listeCell = enumCellules.CreateEnumeration
Do While listeCell.hasMoreElements
  uneCellule = listeCell.nextElement
  uneCellule.String = UCase(uneCellule.String)
Loop
End Sub
```

Nous utilisons la méthode `queryContentCells` de la sélection. Cette sélection peut être une zone de cellules, éventuellement réduite à une seule cellule, ou un ensemble de zones sélectionnées par l'utilisateur. La méthode `queryContentCells` utilise les mêmes indicateurs `CellFlags` que nous avons vus au tableau 12-14. Ici aussi, on peut additionner les valeurs pour retenir les cellules correspondant à un ou plusieurs critère(s).

Nous obtenons l'objet `selectionCell`. Sa méthode `getCells` renvoie un objet contenant les références des cellules et à partir duquel nous créons un objet permettant de recenser les cellules une à une (l'ordre d'obtention des cellules ne peut être prédit). Il ne nous reste plus qu'à effectuer le traitement sur chaque cellule.

Le même principe pourrait, par exemple, rechercher et traiter toutes les cellules non vides d'une zone.

Fonctions mathématiques sur une zone de cellules

L'API OpenOffice.org offre une série de fonctions courantes qui s'appliquent à une zone de cellules et renvoient un résultat numérique de type `Double`. Il suffit d'utiliser la méthode `computeFunction` de l'objet zone de cellules, en donnant en argument une constante nommée (tableau 12-15) qui précise la fonction. Ces constantes sont de la forme :

```
com.sun.star.sheet.GeneralFunction.MAX
```

Tableau 12–15 Fonctions mathématiques de zone de cellules

Constante	Signification
NONE	Aucun calcul n'est effectué.
AUTO	Si tous les contenus sont numériques la fonction SUM est appliquée, sinon la fonction COUNT est appliquée.
SUM	Somme de tous les contenus numériques.
COUNT	Le nombre de tous les contenus, numériques ou non.
AVERAGE	Moyenne de tous les contenus numériques.
MAX	Valeur maximale de tous les contenus numériques.
MIN	Valeur minimale de tous les contenus numériques.
PRODUCT	Produit de tous les contenus numériques.
COUNTNUMS	Nombre des contenus numériques.
STDEV	Déviation standard basée sur un échantillon.
STDEVP	Déviation standard basée sur la population entière
VAR	Variance calculée sur un échantillon.
VARP	Variance calculées sur la population entière.

L'utilisation est très simple :

```
rem Code12-03.sxc   bibli : Fonctions Module4
Option Explicit

Sub FonctionAPIdeZone()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim fonc As Integer, resu As Double
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Fonctions")
maZone = maFeuille.getCellRangeByName("B11:C13")

fonc = com.sun.star.sheet.GeneralFunction.MAX
resu = maZone.computeFunction(fonc)
print "Résultat : " & resu
End Sub
```

Lire et écrire les données d'un tableau

Il est assez pénible de lire et écrire dans plusieurs cellules successivement. Or dans bien des cas, on doit effectuer des traitements sur toutes les cellules d'une zone.

La propriété `DataArray` de l'objet zone de cellules permettent de copier les valeurs de la zone vers une variable tableau de Basic, afin d'effectuer divers traitements, puis de recopier les valeurs du tableau vers la zone de cellules.

Comme exemple, nous allons supposer que la feuille 2 contient en colonnes B et C des nombres représentant des années. Pour des raisons historiques, les années du XX^{e} siècle ont d'abord été écrites à deux chiffres, puis on est passé à quatre chiffres quand on s'est approché de l'an 2000, ce qui fait un tableau incohérent (imaginez le problème avec plusieurs centaines de lignes). Nous allons convertir les dates à deux chiffres en dates du XX^{e} siècle. La macro va utiliser les particularités des variables de type `Variant` (revoir dans le chapitre 5 les tableaux irréguliers). De plus, l'utilisation de `Variant` nous donne une solution plus générale car elle pourrait traiter des cellules contenant du texte.

```
rem Code12-03.sxc   bibli : Calculs Module1
Option Explicit

Sub ConvertirAnnees2chiffres()
Dim monDocument As Object, lesFeuilles As Object
Dim  As Object, maZone As Object
Dim lesAns As Variant, uneLigne As Variant
Dim ligne As Long, colonne As Long
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille2")
maZone = maFeuille.getCellRangeByName("B2:C11")

lesAns = maZone.DataArray
for ligne = LBound(lesAns) to UBound(lesAns)
  uneLigne = lesAns(ligne)
  for colonne = LBound(uneLigne) to UBound(uneLigne)
    if uneLigne(colonne) < 100 then
      uneLigne(colonne) = uneLigne(colonne) +1900
    end if
  next colonne
  lesAns(ligne) = uneLigne
next ligne
maZone.DataArray = lesAns
End Sub
```

La variable `lesAns` reçoit les valeurs des cellules de la zone B2 à C11. Elle est ainsi transformée par l'affectation en un tableau à deux dimensions dont la première est le numéro de ligne et la deuxième le numéro de colonne relativement à la zone de cellules. Pour explorer le tableau, nous utilisons deux index, pour lesquels les fonctions `LBound` et `UBound` donnent les valeurs extrêmes. Il est nécessaire d'utiliser une variable intermédiaire pour lire et modifier chaque élément du tableau.

La macro ne modifie pas les cellules vides car dans ces cas, la comparaison à la valeur 100 est évaluée à `False`... ce qui nous arrange bien !

Cette structure à deux boucles imbriquées fonctionne aussi pour une zone réduite à une seule ligne ou une seule colonne.

Un autre usage de `DataArray` est de recopier les valeurs d'une zone dans une autre zone.

```
zoneCible.DataArray = zoneSource.DataArray
```

De manière similaire, la propriété `FormulaArray` permet de transférer les formules d'une zone vers un tableau et inversement. Attention, la fonction de recopie ne modifiera pas les références des formules !

```
zoneCible.FormulaArray = zoneSource.FormulaArray
```

Coller uniquement les valeurs et non le format

Le but est de recopier d'un endroit vers un autre des zones pouvant, comme ici, être situées sur des feuilles différentes d'un même document Calc. On utilisera de nouveau la propriété `DataArray`. La macro ci-après fonctionne aussi bien pour des valeurs numériques que textuelles. Si une cellule source contient une formule, la valeur calculée de la formule est recopiée.

```
Dim lesFeuilles As Object, zone_src As Object, zone_dest As Object
lesFeuilles = thisComponent.Sheets
zone_src  = lesFeuilles(1).getCellRangeByName("C3:D4")
zone_dest = lesFeuilles(0).getCellRangeByName("F10:G11")
zone_dest.DataArray = zone_src.DataArray
```

Déplacer ou recopier des cellules avec références

Le principal intérêt d'un tableur est sa faculté à déplacer ou recopier intelligemment des cellules contenant des références à d'autres cellules. Par exemple, si vous déplacez une cellule référencée par une autre, la formule de la deuxième cellule sera modifiée automatiquement pour pointer sur le nouvel emplacement de la première. Si vous copiez la deuxième à un autre emplacement, la copie pointera sur une cellule ayant la même position relative. Si une référence comporte des caractères comme `$A3` ou `A$3` ou `$A$3`, le déplacement ou la copie ne modifient pas l'index préfixé par le dollar.

Déplacer une zone de cellules

L'objet feuille expose la méthode `moveRange` qui utilise deux arguments :

1. l'adresse de la cellule qui se trouve au coin haut gauche de la zone d'arrivée,
2. l'adresse de la zone de départ.

La cellule d'arrivée et la zone de départ peuvent se trouver dans la même feuille ou une feuille différente du même document Calc. Bizarrement, le déplacement s'effectue indifféremment en utilisant la méthode `moveRange` de la feuille de départ, de la feuille d'arrivée, ou de toute autre feuille du classeur.

Dans cet exemple et les suivants, nous utiliserons par convention des noms commençant par `c` pour une cellule, `z` pour une zone, `f` pour une feuille.

```
rem Code12-03.sxc   bibli : References Module1
Option Explicit

Sub DeplacerZone()
Dim monDocument As Object, lesFeuilles As Object
Dim fDepart As Object, fArriv As Object
Dim zDepart As Object, cArriv As Object

monDocument = thisComponent
lesFeuilles = monDocument.Sheets

fDepart = lesFeuilles.getByName("Refs")
zDepart = fDepart.getCellRangeByName("A2:B3")
' les feuilles départ et arrivée peuvent être différentes
fArriv = lesFeuilles.getByName("Arrivée")
cArriv = fArriv.getCellRangeByName("F8")
fArriv.moveRange(cArriv.CellAddress, zDepart.RangeAddress)
End Sub
```

Recopier une zone de cellules

La recopie s'effectue exactement de la même manière que pour un déplacement, à l'aide de la méthode `copyRange`, qui utilise les mêmes arguments que `moveRange`.

Nous ne présenterons qu'une partie de l'exemple, que vous pourrez retrouver dans le zip téléchargeable :

```
rem Code12-03.sxc   bibli : References Module2
Option Explicit

Sub CopierZone()
rem  --- tout est identique jusqu'ici --- >
fArriv.copyRange(cArriv.CellAddress, zDepart.RangeAddress)
End Sub
```

Recopier une cellule dans une zone

Comme vous l'avez remarqué, nous ne savons que recopier une zone. Comment copier *une cellule* vers toute une zone, en mettant à jour ses références ?

Dans un premier exemple, nous souhaitons recopier une cellule vers une zone colonne. Nous commençons par copier la cellule dans la première cellule de la zone destinataire. Dans cette opération, la cellule de départ est assimilée à une zone d'une cellule. Puis, nous utilisons la méthode `fillAuto` de la zone destinataire, qui recopie intelligemment la première cellule sur une colonne, de haut en bas.

```
rem Code12-03.sxc   bibli : References Module3
Option Explicit

Sub Copier1CelluleDansZone()
Dim monDocument As Object, lesFeuilles As Object
Dim fDepart As Object, fArriv As Object
Dim cDepart As Object, cArriv As Object, zArriv As Object
Dim direction As Integer
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

fDepart = lesFeuilles.getByName("Refs")
cDepart = fDepart.getCellRangeByName("B3")
' commencer par copier une cellule
fArriv = lesFeuilles.getByName("Arrivée")
cArriv = fArriv.getCellRangeByName("C2")
fArriv.copyRange(cArriv.CellAddress, cDepart.RangeAddress)
' multiplier la cellule à l'arrivée
zArriv = fArriv.getCellRangeByName("C2:C32")
direction = com.sun.star.sheet.FillDirection.TO_BOTTOM
zArriv.fillAuto(direction, 1)
End Sub
```

Nous aurions pu recopier d'abord la cellule en bas de la zone, et compléter de bas en haut :

```
direction = com.sun.star.sheet.FillDirection.TO_TOP
```

Pour une zone d'arrivée horizontale, `fillAuto` peut compléter de gauche à droite :

```
direction = com.sun.star.sheet.FillDirection.TO_RIGHT
```

ou de droite à gauche :

```
direction = com.sun.star.sheet.FillDirection.TO_LEFT
```

Le deuxième exemple consiste à recopier une zone horizontale en l'étalant sur une zone rectangulaire. Le principe est identique et `fillAuto` recopiera la zone horizontale, ici en partant du bas.

```
rem Code12-03.sxc   bibli : References Module4
Option Explicit
```

```
Sub CopierZone1DansZone2()
Dim monDocument As Object, lesFeuilles As Object
Dim fDepart As Object, fArriv As Object
Dim zDepart As Object, cArriv As Object, zArriv As Object
Dim direction As Integer
monDocument = thisComponent
lesFeuilles = monDocument.Sheets

fDepart = lesFeuilles.getByName("Refs")
zDepart = fDepart.getCellRangeByName("D13:E13")
' commencer par copier la zone horizontale
fArriv = lesFeuilles.getByName("Arrivée")
cArriv = fArriv.getCellRangeByName("C9")
fArriv.copyRange(cArriv.CellAddress, zDepart.RangeAddress)
' multiplier la zone à l'arrivée, en remontant
zArriv = fArriv.getCellRangeByName("C3:D9")
direction = com.sun.star.sheet.FillDirection.TO_TOP
zArriv.fillAuto(direction, 1)
End Sub
```

La méthode `fillAuto` (et une autre méthode, `fillSeries`) peut remplir des cellules successives avec une suite de valeurs. Ces possibilités sont inutiles pour nous, car il est bien plus simple et plus puissant de créer un tableau Basic et de le remplir avec une boucle, puis de le recopier dans la feuille avec la propriété `DataArray` d'une zone de cellules.

Recopier une formule dans une zone

Lorsque nous avons décrit plus haut les zones nommées, nous avons signalé qu'elles permettent de recopier une formule avec mise à jour de ses références. Si nous voulons recopier intelligemment une formule qui se trouve dans une cellule vers une zone cible, une solution est de définir une zone nommée qui recueille la formule.

L'exemple qui suit suppose, dans la feuille `Refs`, une cellule en `B23` contenant une formule calculée sur d'autres cellules de la colonne. Le but est de recopier cette formule dans les cellules `H33` à `J33`.

```
rem Code12-03.sxc   bibli : References Module5
Option Explicit

Sub RecopierFormule()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim lesZonesNom As Object, x As Long, cellCible As Object
Const unNom = "x_001h" ' un nom probablement pas utilisé!
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
lesZonesNom = monDocument.NamedRanges
maFeuille = lesFeuilles.getByName("Refs")
maCellule = maFeuille.getCellRangeByName("B23")
```

```
if lesZonesNom.hasByName(unNom) then ' vérifier quand même
  MsgBox("Le nom existe de zone existe déjà",16)
  exit sub
end if
lesZonesNom.addNewByName(unNom, maCellule.Formula, _
                                maCellule.CellAddress,0)
for x = 7 to 9 ' colonnes H à J
  cellCible = maFeuille.getCellByPosition(x,32) ' ligne 33
  cellCible.Formula = "=" & unNom ' recopie "intelligente"
next
End Sub
```

Il est nécessaire de trouver un nom de zone qui ne soit pas déjà utilisé. La formule recopiée utilisera ce nom, qu'il faudra garder pour qu'elle fonctionne. Nous créons la zone nommée en prenant pour cellule de référence la cellule contenant la formule d'origine. Puis nous recopions la formule dans les cellules cibles.

> Les cellules cibles pourraient se trouver dans une autre feuille, ou même dans un autre document Calc (la zone nommée doit alors être définie dans le document cible).

Le code employé a été choisi pour sa simplicité, au détriment de l'adaptabilité. Vous remarquerez cependant que cette solution est indépendante du contenu de la formule.

Rechercher, remplacer

Les mécanismes de recherche dans un document Calc utilisent un objet « descripteur de recherche ». Nous allons l'utiliser dans un premier exemple qui consiste à trouver dans une feuille toutes les occurrences d'un terme dans les cellules texte. Chaque cellule trouvée est ensuite mise en exergue avec un arrière-plan coloré.

```
rem Code12-04.sxc   bibli : Rechercher Module1
Option Explicit

Sub TrouverToutdans1Feuille()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
Dim jeCherche As Object, trouv As Variant
Dim x As Long
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
jeCherche = maFeuille.createSearchDescriptor
with jeCherche
  .SearchString = "Maurice"
  ' rechercher les cellules contenant au moins ce texte
  .SearchWords = false
```

```
end with
trouv = maFeuille.findAll(jeCherche)
print "Nombre d'occurrences : " & trouv.Count
for x = 0 to trouv.Count -1
  trouv(x).CellBackColor = RGB(255,200,255)
next
End Sub
```

Un descripteur de recherche est obtenu avec la méthode `createSearchDescriptor` de l'objet feuille. Ce descripteur comporte plusieurs éléments à remplir avant de lancer la recherche. L'objet feuille fournit la fonction `findAll` qui a pour argument l'objet descripteur de recherche et renvoie un objet conteneur contenant toutes les occurrences trouvées. Leur nombre est disponible dans la propriété `Count` de l'objet conteneur. OOoBasic permet d'obtenir les occurrences par une simple indexation du conteneur ; ce sont des zones de cellules.

La structure Basic `With ... End With` évite de répéter le nom de la variable descripteur sur plusieurs lignes. Sans elle on écrirait :

```
jeCherche.SearchString = "Marseille"
jeCherche.SearchWords = true
```

Vous remarquerez avec l'exemple du zip téléchargeable que les deux occurrences dans les cellules contiguës de la même ligne comptent pour une seule car elles sont présentes dans une seule zone. La fonction « Tout rechercher » dans l'interface utilisateur donne les mêmes résultats.

Notre recherche pourrait facilement être modifiée pour explorer toutes les feuilles du classeur. Il n'est pas nécessaire de changer le descripteur de recherche, seulement exécuter le `findAll` sur chacune des feuilles.

Le descripteur de recherche

Le tableau 12-16 liste les éléments de ce descripteur ; leur utilisation est identique à la fonction de recherche de l'interface utilisateur :

Tableau 12–16 Descripteur de recherche

Élément	Type	Signification
`SearchString`	`String`	La chaîne de caractères à rechercher dans la propriété `String` des cellules.
`SearchBackwards`	`Boolean`	`True` pour faire une recherche à reculons ; par défaut, on recherche dans le sens normal de lecture.
`SearchByRow`	`Boolean`	`True` pour rechercher par lignes. `False` pour rechercher par colonnes (valeur par défaut).

Tableau 12–16 Descripteur de recherche (suite)

Élément	Type	Signification
SearchCaseSensitive	Boolean	True pour distinguer les majuscules des minuscules dans la recherche ; par défaut il n'y a pas de distinction. Quelle que soit la valeur de cette propriété, les caractères accentués sont toujours différenciés des caractères non accentués.
SearchWords	Boolean	True pour rechercher les cellules ne comportant que la chaîne de recherche, sans autre caractère.
SearchType	Integer	Propriété inutilisable.
SearchRegularExpression	Boolean	True pour faire une recherche avec la méthode des expressions régulières ; par défaut, on recherche une simple égalité de chaînes.
SearchStyles	Boolean	True pour rechercher des cellules d'un style donné par SearchString ; le nom du style est celui affiché par le styliste. Par défaut, on cherche un contenu de cellule.
SearchSimilarity	Boolean	True pour rechercher un texte similaire au texte cherché.
SearchSimilarityRelax	Boolean	True pour essayer toute combinaison des trois critères suivants qui permette de retrouver le texte cherché.
SearchSimilarityRemove	Integer	Nombre de caractères à retrancher pour retrouver le texte cherché.
SearchSimilarityAdd	Integer	Nombre de caractères à ajouter pour retrouver le texte cherché.
SearchSimilarityExchange	Integer	Nombre de caractères à changer pour retrouver le texte cherché.

Limiter le champ de la recherche

Nous allons maintenant refaire la recherche, en se limitant à une zone de cellules dans la feuille. L'objet zone de cellules possède les mêmes méthodes, donc les changements sont minimes.

```
rem Code12-04.sxc   bibli : Rechercher Module2
Option Explicit

Sub TrouverToutdans1Zone()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim jeCherche As Object, trouv As Variant
Dim x As Long
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
maZone = maFeuille.getCellRangeByName("B70:C100")
jeCherche = maZone.createSearchDescriptor
```

```
with jeCherche
  .SearchString = "Maurice"
  ' rechercher les cellules contenant au moins ce texte
  .SearchWords = false
end with
trouv = maZone.findAll(jeCherche)
print "Nombre d'occurrences : " & trouv.Count
for x = 0 to trouv.Count -1
  trouv(x).CellBackColor = RGB(255,200,255)
next
End Sub
```

Un autre moyen de recherche consiste à trouver la première occurrence avec la méthode `findFirst` de l'objet document, puis à rechercher les occurrences suivantes avec la méthode `findNext` jusqu'à échec de la recherche. Voici une recherche progressive dans toute la feuille, en explorant ligne par ligne.

```
rem Code12-04.sxc   bibli : Rechercher Module3
Option Explicit

Sub TrouverToutsuccessivement()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
Dim jeCherche As Object, posTrouve As Variant
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
jeCherche = maFeuille.createSearchDescriptor
with jeCherche
  .SearchString = "Otto"
  ' rechercher les cellules contenant au moins ce texte
  .SearchWords = false
  .SearchByRow = true ' rechercher par lignes
end with
posTrouve = maFeuille.findFirst(jeCherche)
Do Until isNull(posTrouve)
  posTrouve.CellBackColor = RGB(255,200,255)
  posTrouve = maFeuille.findNext(posTrouve, jeCherche)
Loop
End Sub
```

Si la recherche ne trouve rien, l'objet `posTrouve` reçoit la valeur `Null` et la boucle n'est pas exécutée. Si elle trouve, `posTrouve` est alors un objet cellule. La recherche est relancée par `findNext`, qui utilise en premier argument l'objet cellule précédemment trouvé.

Ici, nous avons forcé la recherche ligne par ligne, ce qui est visible si vous posez un point d'arrêt sur l'instruction comportant `findNext`.

Comme avec le deuxième exemple, nous pourrions limiter la recherche à une zone de cellules. D'ailleurs, elle pourrait être une zone sélectionnée par l'utilisateur, comme décrit plus haut dans la section sur les sélections visuelles.

Il est possible de sélectionner visuellement la zone trouvée, au lieu de changer la couleur de fond :

```
monDocument.CurrentController.Select(posTrouve)
```

Rechercher pour remplacer

Lors d'une recherche progressive, la zone de texte `posTrouve` sélectionne la cellule trouvée. Nous pourrions alors utiliser cet objet cellule pour analyser le type de contenu de la cellule, ou modifier le contenu ou le format à notre guise. Nous vous laissons faire une telle macro, à titre d'exercice.

Tout remplacer

Dans le cas simple où toutes les occurrences doivent être systématiquement remplacées partout, les objets feuille et zone offrent la fonction `replaceAll`, qui effectue ce travail et renvoie le nombre de remplacements. Elle utilise un descripteur de remplacement, obtenu avec la méthode `createReplaceDescriptor`, qui contient tous les éléments du descripteur de recherche et ajoute `ReplaceString`, de type `String`, qui contient la chaîne de caractères à mettre à la place de celle qu'on recherche.

```
rem Code12-04.sxc   bibli : Remplacer Module1
Option Explicit

Sub RemplacerPartoutdans1Feuille()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
Dim jeCherche As Object, nbrFois As Long
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
jeCherche = maFeuille.createReplaceDescriptor
with jeCherche
  .SearchString = "Maurice"
  .ReplaceString = "Marcel"
  ' rechercher les cellules contenant au moins ce texte
  .SearchWords = false
end with
nbrFois = maFeuille.replaceAll(jeCherche)
print "Nombre de remplacements : " & nbrFois
End Sub
```

La macro suivante modifie toutes les cellules de style `Titre1` pour leur affecter un autre style `Résultat2`.

```
rem Code12-04.sxc   bibli : Remplacer Module2
Option Explicit

Sub RemplacerStylePartoutdans1Feuille()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
Dim jeCherche As Object
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
jeCherche = maFeuille.createReplaceDescriptor
with jeCherche
  .SearchString = "Titre1"
  .ReplaceString = "Résultat2"
  .SearchStyles = true
end with
maFeuille.replaceAll(jeCherche)
End Sub
```

Dans une modification de styles, le nombre de remplacements n'est pas fourni par `replaceAll`.

Trier un tableau

> **ATTENTION Compatibilité**
>
> La version 1.1 d'OpenOffice.org a modifié le mécanisme de tri. Nous ne décrirons que ce dernier, qui ne fonctionne pas sur les versions 1.0.x ou antérieures.

Un objet zone de cellules fournit une méthode de tri (en anglais *Sort*). Nous allons effectuer un tri sur la première colonne du tableau de la feuille `Compositeurs`. Voici l'ensemble du code, que nous expliquerons progressivement.

```
rem Code12-03.sxc   bibli : Calculs Module2
Option Explicit

Sub Trier1Colonne()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim ConfigTri(0) As New com.sun.star.table.TableSortField
Dim DescrTri As Variant
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
maZone = maFeuille.getCellRangeByName("A1:D118")
```

```
With ConfigTri(0)
  .Field = 0 ' colonne A = "Nom, Prénom"
  .IsAscending = true
End With

DescrTri = maZone.createSortDescriptor
setPropVal(DescrTri, "SortFields", ConfigTri())
setPropVal(DescrTri, "IsSortColumns", false)
setPropVal(DescrTri, "CopyOutputData", false)
setPropVal(DescrTri, "IsUserListEnabled", false)
setPropVal(DescrTri, "BindFormatsToContent", false)
setPropVal(DescrTri, "ContainsHeader", true)
maZone.Sort(DescrTri())
End Sub

Sub setPropVal(descr As Variant, _
               nomProp As String, valProp As Variant)
Const Titre = "Tableau de propriétés"
Dim x As Long
for x = 0 to UBound(descr)
  if descr(x).Name = nomProp then
    descr(x).Value = valProp
    Exit Sub
  end if
next
MsgBox("Propriété inconnue : " & nomProp, 16, Titre)
End Sub
```

Nous définissons la zone de tri dans le tableau, en incluant la première ligne qui contient les en-têtes de colonnes. On aurait pu l'exclure.

La méthode de tri a besoin d'un descripteur de tri (notre variable `DescrTri`), qui précise les conditions globales de tri. Parmi celles-ci, nous trouvons l'élément `SortFields`, qui contient un tableau (`Array`) de descripteurs, un par colonne à trier (notre variable `ConfigTri`). Nous allons maintenant décrire chaque descripteur.

Le descripteur de tri par colonne est une structure `com.sun.star.table.TableSortField` composée des éléments détaillés dans le tableau 12-17.

Tableau 12–17 Descripteur de tri par colonne

Élément	Type	Signification
`Field`	`Long`	Rang de la colonne dans la zone de cellules ; valeur 0 pour la première colonne.
`IsAscending`	`Boolean`	`True` pour un tri par ordre croissant.
`IsCaseSensitive`	`Boolean`	`True` pour tenir compte de la casse des caractères
`FieldType`	`Integer`	Type du champ. (non utilisé)
`CollatorLocale`	`Object`	Langue et variante nationale de langue.
`CollatorAlgorithm`	`String`	Méthode de tri.

L'élément `FieldType` n'est pas utilisé car chaque cellule de tableur indique le type de donnée qu'elle contient. Le tri croissant place les cellules numériques avant les cellules de texte.

L'élément `CollatorLocale` est lui-même une structure, décrite plus haut, section « Format de nombre ». Cependant, pour les cas courants (français, anglais, espagnol, allemand...) il n'est pas nécessaire de le remplir. De même, `CollatorAlgorithm` peut être omis car il n'y a actuellement pas de choix possible (la seule valeur possible est : `alphanumeric`).

Le descripteur global de tri est obtenu avec la méthode `createSortDescriptor` de l'objet `maZone`. Ce descripteur est un tableau (`Array`) de structures `PropertyValue`. Chacune de celles-ci comporte un nom et une valeur. Le tableau 12-18 liste ces propriétés. Les noms sont initialisés par `createSortDescriptor`, mais l'ordre des propriétés dans le tableau de structures dépend de l'implémentation. La routine utilitaire `setPropVal` recherche la propriété d'un nom donné et lui affecte la valeur qui est en argument. Cette routine est empruntée à l'annexe B.

Affectez une valeur à chacun des indicateurs booléens du descripteur global, car ils ne sont pas tous initialisés à `False`.

L'élément `IsSortColumns`, au nom particulièrement mal choisi, indique si on trie par colonne (notre cas, le cas courant), ou si on trie des lignes (dans ce cas, le tableau aura des en-têtes de lignes et tout se passe comme si on avait tourné le tableau de 90 degrés). Nous continuons sur notre exemple de tri de colonnes, sachant qu'il suffit de remplacer le terme « ligne » par « colonne » dans le deuxième cas.

L'élément `MaxFieldCount` est pour nous une constante. Dans la version actuelle d'OpenOffice.org, on peut trier sur un maximum de 3 colonnes, comme on le verra dans l'exemple suivant.

Tableau 12–18 Descripteur global de tri

Élément	Type	Signification
`IsSortColumns`	`Boolean`	`False` pour trier par colonne (intervertir les lignes). Valeur par défaut. `True` pour trier par lignes (intervertir les colonnes).
`SortFields`	`Array(Object)`	Tableau des descripteurs de tri par colonne.
`ContainsHeader`	`Boolean`	`True` si la zone contient les en-têtes de colonne.
`BindFormatsToContent`	`Boolean`	`True` pour déplacer les formats de cellules avec les cellules. Valeur par défaut.
`MaxFieldCount`	`Long`	En lecture seule, nombre maximal de colonnes à trier, valeur actuelle = 3.
`CopyOutputData`	`Boolean`	`True` pour mettre le résultat trié dans une autre zone. Valeur par défaut : `False`.

Tableau 12–18 Descripteur global de tri

Élément	Type	Signification
`OutputPosition`	`Object`	Position de la première cellule de la zone à utiliser pour le résultat.
`IsUserListEnabled`	`Boolean`	`True` pour utiliser une liste de tri particulière. Valeur par défaut : `False`.
`UserListIndex`	`Long`	Précise la liste de tri utilisée.

La position `OutputPosition` est obtenue avec la propriété `CellAddress` d'un objet cellule.

Nous ne détaillerons pas le concept de `UserListIndex`. Les tris utilisateurs forment un tableau de chaînes de caractères, mémorisées dans les propriétés du document.

Une fois le descripteur global de tri initialisé, la table est triée avec la méthode `Sort`. Si vous exécutez la macro sur le document fourni dans le zip téléchargeable, vous pourrez voir comment s'effectue le tri avec ou sans prise en compte de la casse. Pour revenir à l'ordre initial de la table, utilisez le bouton Annuler sur la fenêtre Writer, ou exécutez la macro `TrierTableParLignes` qui est dans le module 3 de la même bibliothèque.

Nous allons maintenant trier le même tableau sur trois colonnes successivement. Nous utiliserons un tableau de 3 descripteurs de colonnes, qui sont prises en compte dans le même ordre que ces descripteurs.

```
rem Code12-03.sxc   bibli : Calculs Module3
Option Explicit

Sub Trier3Colonnes()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim ConfigTri(2) As New com.sun.star.table.TableSortField
Dim DescrTri As Variant
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Compositeurs")
maZone = maFeuille.getCellRangeByName("A1:D118")

ConfigTri(0).Field = 2 ' colonne C = "Naissance"
ConfigTri(0).IsAscending = true
ConfigTri(1).Field = 3 ' colonne D = "Décès"
ConfigTri(1).IsAscending = false
ConfigTri(2).Field = 0 ' colonne A = "Nom, Prénom"
ConfigTri(2).IsAscending = true

DescrTri = maZone.createSortDescriptor
setPropVal(DescrTri, "SortFields", ConfigTri())
setPropVal(DescrTri, "IsSortColumns", false)
setPropVal(DescrTri, "CopyOutputData", false)
```

```
setPropVal(DescrTri, "IsUserListEnabled", false)
setPropVal(DescrTri, "BindFormatsToContent", false)
setPropVal(DescrTri, "ContainsHeader", true)
maZone.Sort(DescrTri())
End Sub
```

Le tri sur plusieurs colonnes ne prend en compte qu'une seule valeur pour le choix de la casse et pour le choix du `Locale`.

Le tri de tableau est décrit dans le SDK à l'interface `XSortable` et aux services `SortDescriptor2`, `TableSortDescriptor2` et `SheetSortDescriptor2`. Les tris utilisateurs sont cités dans le service `GlobalSheetSettings`.

Utiliser une fonction de Calc

Basic dispose d'un certain nombre de fonctions, mais Calc en possède de nombreuses autres. Dans vos algorithmes de macros, il est inutile de programmer des fonctions qui existent dans Calc : il suffit de les appeler.

Notre premier exemple va calculer le PPCM (Plus Petit Commun Multiple) de plusieurs nombres. Il utilise une petite fonction que nous appellerons de manière très originale : `PPCM`.

```
rem Code12-03.sxc   bibli : Fonctions Module1
Option Explicit

Sub UtiliserFonctionCalc()
Dim params As Variant
params = Array(35,75,1110)
print PPCM(params())
' appel sans utiliser de variable intermédiaire
print PPCM(Array(17, 525, 357, 76, 54))
End Sub

Function PPCM(liste As Variant) As Long
Dim acceder As Object
acceder = CreateUnoService("com.sun.star.sheet.FunctionAccess")
PPCM = acceder.callFunction("LCM", liste)
End Function
```

Notre fonction `PPCM` obtient un objet lui permettant d'utiliser le service d'accès aux fonctions Calc. La méthode `callFunction` de ce service appelle une fonction Calc (son nom anglais est en premier argument, les arguments de la fonction sont en deuxième argument), puis renvoie le résultat simplement comme résultat de notre fonction.

La routine principale montre deux exemples d'appels. Le premier utilise la fonction Basic `Array()` pour créer et initialiser facilement un tableau unidimensionnel qui contient les valeurs numériques dont on recherche le PPCM. Ce tableau est ensuite transmis à notre fonction PPCM, qui s'empresse de le transmettre à la fonction Calc.

Le deuxième appel est similaire au premier, mais utilise directement le résultat de la fonction `Array` sans variable intermédiaire.

De ces appels, il faut retenir que le deuxième argument de `callFunction` est une liste (un tableau unidimensionnel). Chaque élément de la liste est *un* argument pour la fonction Calc. Un argument de fonction Calc peut admettre une valeur ou une série de valeurs ; la série de valeurs se traduit en un tableau uni ou bidimensionnel.

À RETENIR

Vous avez sans doute remarqué que l'exemple n'utilise aucune référence à une feuille ou à un document Calc. Effectivement, il fonctionne aussi bien dans un document Writer, Draw, ou Impress.

Reste un problème : comment connaître le nom anglais d'une fonction Calc ? Avec une petite macro qui utilise la propriété `FormulaLocal` d'une cellule. Ici, la formule écrite dans la cellule `B2` de la feuille `Fonctions` est affichée en dessous, en français et en anglais.

```
rem Code12-03.sxc   bibli : Fonctions Module2
Option Explicit

Sub TraduireFormule()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
Dim formule As Object, langueE As Object, langueF As Object
monDocument = ThisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Fonctions")
formule = maFeuille.getCellRangeByName("B2")
langueF = maFeuille.getCellRangeByName("B3")
langueE = maFeuille.getCellRangeByName("B4")
langueF.String = formule.FormulaLocal
langueE.String = formule.Formula
End Sub
```

Le deuxième exemple montre comment appeler la fonction `TAUX.EFFECTIF` qui nécessite deux arguments. La variable `params` est définie comme un tableau unidimensionnel à deux éléments, chacun recevant un argument.

```
rem Code12-03.sxc   bibli : Fonctions Module3
Option Explicit

Sub Fonction2Arguments()
Dim acceder As Object, resu As Double
Dim params(1) ' deux arguments à transmettre
acceder = CreateUnoService("com.sun.star.sheet.FunctionAccess")
params(0) = 9.75/100 ' taux nominal
params(1) = 12 ' nombre de paiements annuels
```

```
resu = acceder.callFunction("EFFECTIVE", params())
print "Taux effectif : " & Format(resu, "##.##%")
End Sub
```

BOGUE Fonction Format

Dans OpenOffice.org 1.1, la fonction `Format` est minée par un grand nombre de bogues (pour notre exemple, ce n'est pas important). Évitez de l'utiliser. La fonction devrait être corrigée pour la version 2.0.

Créer une nouvelle fonction pour Calc

Si les nombreuses fonctions de Calc ne vous suffisent pas, écrivez en Basic une fonction qui réalise ce dont vous avez besoin. Cette fonction Basic doit être accessible par Calc. Cela signifie que la bibliothèque dans laquelle se trouve votre fonction doit être chargée. Plusieurs cas sont possibles selon la bibliothèque :

- Une fonction dans la bibliothèque `Standard` de `soffice` est disponible pour tout document Calc.
- Une fonction dans la bibliothèque `Standard` d'un document Calc est disponible dans ce document seulement.
- Une fonction dans une bibliothèque *quelconque* de `soffice` est disponible pour tout document Calc, une fois la bibliothèque chargée (le chapitre 7 indique comment charger une bibliothèque).
- Une fonction dans une bibliothèque quelconque d'un document Calc est disponible dans ce document seulement, une fois la bibliothèque chargée.

Vous devez bien comprendre ce qui précède, sinon votre fonction sera disponible un jour et pas l'autre, en fonction des manipulations effectuées, ce qui vous amènera au bord de la crise de nerfs.

Nous allons commencer simplement, avec une fonction qui remplace un nombre à deux chiffres par une année du XX^{e} siècle. Si le nombre est négatif ou supérieur à 2199, on veut déclencher une erreur.

```
rem Code12-03.sxc   bibli : Standard Module1
Option Explicit

' nouvelle fonction pour Calc
Function Annee2( arg As Long) As Variant
if (arg<0) or (arg>2099) then
  Annee2 = "Année ?"
elseif arg<100 then
  Annee2 = 1900 +arg
```

```
else
  Annee2 = arg
end if
End Function
```

La fonction renvoie normalement un nombre, sauf en cas d'argument incorrect où elle renvoie un texte d'erreur. Suivant la formule contenue dans la cellule, soit le texte sera affiché, soit le mélange de texte et de valeur numérique provoquera une erreur. Votre fonction peut comporter plusieurs arguments, comme dans cet exemple qui calcule un volume à partir de trois dimensions :

```
rem Code12-03.sxc   bibli : Standard Module2
Option Explicit

' nouvelle fonction pour Calc
Function Volume(arg1 As Double, arg2 As Double, _
                arg3 As Double) As Double
Volume = arg1 *arg2 *arg3
End Function
```

Enfin, si un argument utilise les valeurs d'une zone de cellules, il apparaît sous forme d'un tableau à deux dimensions. Ce dernier exemple montre comment obtenir les valeurs successives de la zone de cellules. Il fonctionne pour une zone rectangulaire ou réduite à une dimension.

```
rem Code12-03.sxc   bibli : Standard Module3
Option Explicit

' nouvelle fonction pour Calc
Function maSomme(arg As Variant) As Double
Dim colonne As Long, ligne As Long, resu As Double
resu = 0
for ligne = LBound(arg) to UBound(arg)
  for colonne = LBound(arg, 2) to UBound(arg, 2)
    resu = resu + arg(ligne,colonne)
  next colonne
next ligne
maSomme = resu
End Function
```

LIMITATION **Fonction Calc**

Une fonction Calc sert à renvoyer une valeur qui sera utilisée dans la formule de la cellule appelant cette fonction. Vous ne pouvez pas profiter de cette fonction pour modifier une autre cellule.

Insérer un lien hypertexte

Cette macro insère un lien hypertexte dans le texte d'une cellule.

```
rem Code12-09.sxc   bibli : Standard Module1
Option Explicit

Sub InsererLienHyperTexte()
Dim monDocument As Object, maFeuille As Object
Dim maCellule As Object, monCurseur As Object
Dim monHyper As Object

monDocument = ThisComponent
maFeuille = monDocument.Sheets.getByName("HyperTexte")
maCellule = maFeuille.getCellRangeByName("B2")
monHyper = monDocument.createInstance("com.sun.star.text.TextField.URL")
monHyper.URL = "http://fr.openoffice.org"
monHyper.Representation = "OpenOffice.org France"
monCurseur = maCellule.createTextCursor
monCurseur.gotoEnd(false)
maCellule.insertTextContent(monCurseur, monHyper, false)
End Sub
```

Configuration d'affichage du document

Ces propriétés du document Calc (tableau 12-19), qui existent dans l'interface utilisateur principalement au niveau Outils > Options > Classeur > Affichage, sont accessibles à partir de l'objet contrôleur du document. Elles sont, pour certaines, mémorisées à la sauvegarde du document.

```
rem Code12-06.sxc   bibli : Config Module1
Option Explicit

Sub ConfigDocCalc()
Dim monDocument As Object, leControleur As Object
monDocument = thisComponent
' voir des valeurs pour exemple dans la feuille 2
leControleur = monDocument.CurrentController
With leControleur
  .ShowZeroValues = MsgBox("ShowZeroValues ?", 4) = 6
  .ShowNotes = MsgBox("ShowNotes ?", 4) = 6
  .ShowGrid = MsgBox("ShowGrid ?", 4) = 6
  .ShowPageBreaks = MsgBox("ShowPageBreaks ?", 4) = 6
  .HasColumnRowHeaders = MsgBox("HasColumnRowHeaders ?", 4) = 6
  .ShowFormulas = MsgBox("ShowFormulas ?", 4) = 6
end With
End Sub
```

Tableau 12–19 Propriétés de configuration de Calc

Propriété	Type	Signification
ShowZeroValues	Boolean	True pour afficher les valeurs nulles.
ShowFormulas	Boolean	True pour afficher le texte des formules.
ShowNotes	Boolean	True pour afficher l'indicateur de note dans les cellules.
ShowGrid	Boolean	True pour afficher la grille délimitant les cellules.
IsValueHighlightingEnabled	Boolean	True pour utiliser des couleurs différentes à l'affichage des textes, valeurs, formules.
HasVerticalScrollBar	Boolean	True pour afficher l'ascenseur vertical.
HasHorizontalScrollBar	Boolean	True pour afficher l'ascenseur horizontal.
HasSheetTabs	Boolean	True pour afficher les onglets des feuilles.
IsOutlineSymbolsSet	Boolean	True pour afficher les symboles de plan.
HasColumnRowHeaders	Boolean	True pour afficher les en-têtes de lignes et colonnes.
ShowPageBreaks	Boolean	True pour afficher les sauts de page.
ShowHelpLines	Boolean	True pour afficher des repères pendant le déplacement d'une forme.
ShowAnchor	Boolean	True pour afficher l'ancre.
SolidHandles	Boolean	True pour afficher une marque visible pendant le déplacement d'une forme.
ShowObjects	Boolean	True pour afficher les objets.
ShowCharts	Boolean	True pour afficher les diagrammes
ShowDrawing	Boolean	True pour afficher les formes.
HideSpellMarks	Boolean	True pour cacher les marques du correcteur d'orthographe.
ZoomType	Integer	Facteur de zoom ; constante nommée, voir chapitre 10.
ZoomValue	Integer	Valeur du zoom, voir chapitre 10.
GridColor	Long	Couleur du trait de la grille.

Imprimer

Le mécanisme général d'impression est décrit au chapitre 10. Nous traiterons ici des particularités de Calc.

Zones d'impression

Les zones à imprimer (interface utilisateur Format > Zones d'impression) sont définies au niveau d'une feuille dans la propriété PrintAreas. Elle reçoit un tableau de coordonnées de zones d'impression. Ces coordonnées sont obtenues avec la propriété RangeAddress de chaque zone. L'impression se fera à raison d'une page (au moins) par zone.

```
rem Code10-02.sxc   bibli : Imprimer Module4
Option Explicit

Sub ImprimerZones()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maZone As Object
Dim Props() As New com.sun.star.beans.PropertyValue
Dim adrZones(1) As New com.sun.star.table.CellRangeAddress
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Météo")
maZone = maFeuille.getCellRangeByName("B4:E6")
adrZones(0) = maZone.RangeAddress
maZone = maFeuille.getCellRangeByName("G6:K8")
adrZones(1) = maZone.RangeAddress
maFeuille.PrintAreas = adrZones()
if MsgBox("Imprimer ?", 36) = 6 then
  ' Props() est vide, pas d'option particulière
  monDocument.Print(Props()) ' une page par zone
end if
End Sub
```

Les zones d'impression sont visibles avec le menu Format > Zones d'impression > Éditer... La lecture de la propriété `PrintAreas` donne un tableau de coordonnées de zones d'impression.

Répéter les en-têtes

Le tableau 12-20 liste les propriétés de la feuille qui gouvernent l'impression des en-têtes sur les pages successives. Comme ce sont des propriétés de feuille, on ne doit alors imprimer qu'une seule zone dans la feuille.

Tableau 12–20 Options d'impression

Propriété	Type	Signification
`PrintTitleColumns`	`Boolean`	`True` si l'en-tête des colonnes est imprimé sur chaque page.
`TitleColumns`	`Object`	Coordonnées de la zone d'en-tête des colonnes.
`PrintTitleRows`	`Boolean`	`True` si l'en-tête des lignes est imprimé sur chaque page.
`TitleRows`	`Object`	Coordonnées de la zone d'en-tête des lignes.

Pour répéter l'impression de l'en-tête des colonnes, il suffit d'ajouter des instructions comme :

```
maZone = maFeuille.getCellRangeByName("B4:E4")
maFeuille.TitleColumns = maZone.RangeAddress
maFeuille.PrintTitleColumns = True
```

Les diagrammes

La grande richesse des diagrammes est bien représentée par l'interface utilisateur, tant par l'outil AutoFormat de diagramme que par les multiples et parfois complexes possibilités de modifications manuelles. S'il est possible d'insérer par macro un diagramme dans une feuille de tableur, ajuster tous les paramètres nécessaires représente une tâche considérable et peu productive. Il est bien plus simple d'utiliser un document existant (ou un modèle) dans lequel les diagrammes ont été créés avec l'interface utilisateur.

Obtenir un diagramme existant

Chaque feuille d'un document Calc peut contenir plusieurs diagrammes. La collection des diagrammes d'une feuille est exposée par l'objet `Charts` inclus dans celle-ci. Chaque diagramme possède un nom, qui n'est pas accessible à l'utilisateur, et un objet inclus, `embeddedObject`, qui contient les caractéristiques du diagramme en tant que dessin.

Un diagramme est en effet un dessin. Or, chaque page de Calc contient une page de dessin, `Drawpage`, qui est la collection des dessins de la feuille. Chaque dessin possède un nom, modifiable par l'utilisateur : dans le cas d'un diagramme, le sélectionner, faire un clic droit et choisir `Nommer l'objet...` . Un dessin de diagramme supporte le service `OLE2shape` et expose l'objet `Model` qui n'est autre que l'objet inclus déjà vu. La figure 12-1 résume ces relations.

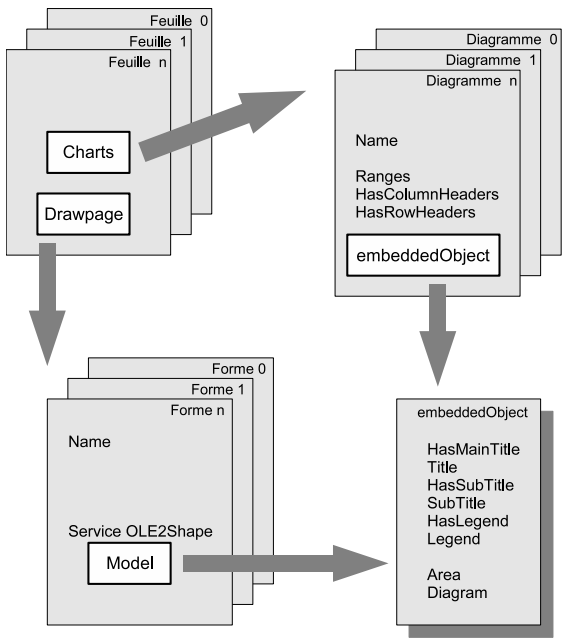


Figure 12–1
Les diagrammes dans Calc

Il existe plusieurs moyens d'accéder à l'objet inclus. Si vous êtes certain qu'il n'existe qu'un seul diagramme dans la feuille, cette instruction suffit :

```
Dim monDiag As Object
monDiag = maFeuille.Charts(0).embeddedObject
```

En effet, comme toute collection, celle-ci est accessible avec OOoBasic par simple indexation, le nombre de diagrammes étant fourni par la propriété `Count` de l'objet `Charts`.

Si la feuille peut comporter plusieurs diagrammes, il est risqué de se fier à l'index, car la position du diagramme dépend de l'ordre de création. On peut éventuellement explorer chaque diagramme en recherchant un nom significatif de titre ou sous-titre. Si vous connaissez le nom interne, l'objet est accessible ainsi :

```
Dim monDiag As Object
monDiag = maFeuille.Charts.getByName("Nom_xxx").embeddedObject
```

En réalité, il est peu probable que vous maîtrisiez le nom interne. En revanche, vous avez toute latitude de nommer chacun de vos diagrammes avec l'interface utilisateur. Nous allons retrouver l'objet diagramme à partir de ce nom, grâce à la routine utilitaire `FindChartByObjName` de l'annexe B. Pour la démonstration, nous vérifierons qu'il s'agit bien d'un objet de la collection `Charts` de la feuille.

Le document du zip téléchargeable comporte un diagramme nommé `CapaDD` dans la feuille `DiagLine`.

```
rem Code12-08.sxc   bibli : Diagrammes Module1
Option Explicit

Sub RetrouverDiagramme()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
Dim diag1 As Object, diag2 As Object, x As Long
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("DiagLigne")

' rechercher l'objet OLE par son nom
diag1 = FindChartByObjName(maFeuille, "CapaDD")
if IsNull(diag1) then
  print "Le diagramme n'existe pas !"
else ' retrouver le même diagramme dans la collection
  for x = 0 to maFeuille.Charts.Count -1 ' chaque diagramme
    diag2 = maFeuille.Charts(x).embeddedObject
    if EqualUnoObjects(diag1, diag2) then
      print "Rang du diagramme dans la collection : " & x
      print "Nom du diagramme : " & maFeuille.Charts(x).Name
    end if
```

```
  next
end if
End Sub
```

La fonction utilitaire `FindChartByObjName` renvoie un objet diagramme, ou `Null` en cas d'échec. Pour retrouver le diagramme dans la collection, nous utilisons la fonction Basic `EqualUnoObjects` qui renvoie `True` si les deux variables désignent le même objet. Les instructions `print` finales affichent le rang du diagramme trouvé et son nom interne.

Les propriétés d'un diagramme

Le tableau 12-21 liste les principales propriétés de l'objet inclus, que nous désignerons par `monDiag` dans l'exemple qui suit.

Tableau 12–21 Propriétés de l'objet inclus dans un diagramme

Propriété	Type	Signification
`HasMainTitle`	`Boolean`	`True` si un titre est utilisé.
`Title`	`Object`	Titre du diagramme.
`HasSubTitle`	`Boolean`	`True` si un sous-titre est utilisé.
`SubTitle`	`Object`	Sous-titre du diagramme.
`HasLegend`	`Boolean`	`True` si la légende est affichée.
`Legend`	`Object`	Légende du diagramme.
`Area`	`Object`	Arrière-plan du diagramme.
`Diagram`	`Object`	Le dessin du diagramme.

Chacun de ces objets possède de multiples propriétés ou objets, que nous n'allons pas développer ici car elles sont réglables directement avec l'interface utilisateur.

ATTENTION

Certaines caractéristiques – comme les dimensions de la légende – sont calculées automatiquement par OpenOffice.org en fonction du contenu. N'espérez pas les modifier via l'API.

Si vous souhaitez étudier ce sujet, utilisez l'outil Xray (voir annexe C) pour analyser les propriétés et méthodes de `monDiag` et afficher la documentation de chaque objet. Continuez l'analyse sur les sous-objets. À titre d'exemple, nous allons affecter au titre d'un diagramme le texte de la cellule `A1` de la même feuille et changer la couleur des caractères.

```
rem Code12-08.sxc   bibli : Diagrammes Module2
Option Explicit
```

```
Sub ChangerTitreDiagramme()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maCellule As Object
Dim monDiag As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("DiagLigne")
maCellule = maFeuille.getCellRangeByName("A1")
' rechercher l'objet OLE par son nom
monDiag = FindChartByObjName(maFeuille, "CapaDD")
if IsNull(monDiag) then
  print "Le diagramme n'existe pas !"
else
  maCellule = maFeuille.getCellRangeByName("A1")
  monDiag.Title.CharColor = RGB(120, 0, 0)
  monDiag.Title.String = maCellule.String
end if
End Sub
```

Les styles

Un document Calc comporte des styles de cellule et de page. Il est de loin préférable de définir vos styles avec le styliste dans un document qui vous servira de modèle. Néanmoins, vous pouvez avoir à faire quelques modifications d'un style existant, ce que nous allons essentiellement décrire.

Trouver les styles

Les styles sont regroupés en familles, par exemple la famille des styles de cellule. Dans chaque famille, on trouve une collection de styles. La macro suivante liste tous les styles d'un document Calc.

```
rem Code12-07.sxc   bibli : ModifStyles Module1
Option Explicit

Sub ListerStyles()
Dim monDocument As Object, nomFam As String
Dim lesFamilles As Object, uneFamille As Object
Dim styleX As Object, liste As String, cr As String
Dim f As Long, x As Long
cr = chr(13) ' retour à la ligne
monDocument = thisComponent
```

```
lesFamilles = monDocument.StyleFamilies
for f = 0 to lesFamilles.Count -1 ' chaque famille
  nomFam = lesFamilles.ElementNames(f)
  uneFamille = lesFamilles.getByName(nomFam)
  liste = "--- Famille : " & nomFam & cr
  for x = 0 to uneFamille.Count -1 ' chaque style dans famille
    styleX = uneFamille(x)
    liste = liste & styleX.Name & " = " & _
                    styleX.DisplayName & cr ' nom localisé
  next x
  MsgBox(liste)
next f
End Sub
```

La propriété `StyleFamilies` de l'objet document donne accès à toutes les familles de styles qui existent dans le document. Le nombre de familles est la propriété `Count` de l'objet `lesFamilles` ; la liste des noms de familles est disponible dans le tableau `ElementNames` de ce même objet.

Bien qu'en Basic nous puissions obtenir successivement chaque famille par une simple indexation de l'objet `lesFamilles`, cet ordre n'est pas forcément le même que dans `ElementNames`. C'est pourquoi nous obtenons chaque famille en utilisant la fonction `getByName` de l'objet `lesFamilles`.

Le nombre de styles dans une famille est indiqué par la propriété `Count` de celle-ci. Nous obtenons chaque style par une simple indexation. Le nom du style est obtenu avec sa propriété `Name`. Les styles prédéfinis en standard ont un nom interne anglais. Le nom localisé est récupéré grâce à la propriété `DisplayName`.

Récupérer ou supprimer un style existant

Si vous connaissez le nom du style à modifier, et évidemment le nom de la famille de styles, il sera facile de récupérer le style car les objets disposent d'un accès par le nom. Les familles de styles ont pour nom :

```
CellStyles        PageStyles
```

Cette macro récupère le style de page `Standard` :

```
Dim nomStyleMaPage As String, StyleMaPage As Object
Dim lesStylesPage As Object
lesStylesPage = monDocument.StyleFamilies.getByName(_
                                          "PageStyles")
StyleMaPage = lesStylesPage.getByName("Standard")
```

Notez que `getByName` fonctionne pour un nom de style anglais ou localisé. Pour tester si un style existe dans une famille, on utilise la fonction `HasByName` :

```
if lesStylesPage.hasByName("Report") then
  ' faire un traitement
end if
```

Pour supprimer un style personnel existant, on utilise la méthode `removeByName` :

```
lesStylesPage.removeByName("monStyle")
```

À défaut de créer votre document à partir d'un modèle comportant les styles souhaités (la solution la plus simple), vous pouvez copier dans votre document des styles existant dans un autre document. La méthode `loadStylesFromURL` de l'objet `StyleFamilies` copie par défaut l'ensemble des styles et écrase les styles existants du même nom. Cet exemple ne charge que les styles de page et ne modifie pas les styles existants.

```
rem Code12-07.sxc   bibli : ModifStyles Module2
Option Explicit

Sub ChargerStyles()
Dim monDocument As Object
Dim refDoc As String
Dim options(1) As New com.sun.star.beans.PropertyValue
options(0).Name = "LoadPageStyles"
options(0).Value = True ' ne charger que les styles de page
options(1).Name = "OverwriteStyles"
options(1).Value = False ' ne pas écraser les styles existants
refDoc = convertToURL("C:\Mes modeles\MaReference.stc")
monDocument = ThisComponent
monDocument.StyleFamilies.loadStylesFromURL(refDoc, options())
End Sub
```

On peut définir plusieurs options de chargement, listées au tableau 12-22.

Tableau 12–22 Propriétés optionnelles de chargement de style

Propriété	Type	Signification
`LoadPageStyles`	`Boolean`	`True` pour charger les styles de page.
`LoadCellStyles`	`Boolean`	`True` pour charger les styles de cellule.
`OverwriteStyles`	`Boolean`	`True` pour charger tous les styles dans toutes les familles et écraser tous les styles existants.

Créer un nouveau style

Avec les styles de Calc, on doit d'abord insérer un nouveau style dans sa famille, puis le modifier. La macro ci-dessous crée un nouveau style de cellule, l'initialise à partir d'un style existant, et change une de ses propriétés. Dans le Styliste, basculez éventuellement l'affichage d'une famille de styles à l'autre pour faire apparaître le nouveau style.

```
rem Code12-07.sxc   bibli : ModifStyles Module3
Option Explicit


Sub HeriterStyle()
Dim monDocument As Object
Dim lesFamilles As Object, uneFamille As Object
Dim nouvStyle As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("CellStyles")

nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.CellStyle")
uneFamille.insertByName("Résultat2 en bleu", nouvStyle)
nouvStyle.ParentStyle = "Résultat2" ' hériter d'un style
nouvStyle.CharColor = RGB(0,100,255) ' changer la couleur
End Sub
```

L'héritage d'un style avec la propriété `ParentStyle` ne fonctionne pas pour les styles de page car ils ne sont pas hiérarchisés. On obtient alors un style conforme au style `Standard`, qu'il faudra adapter à ses besoins.

Les propriétés de style

Style de cellule

Dans le style de cellule, nous retrouvons :

- les propriétés globales de la cellule,
- les propriétés de formatage local de la cellule.

Ces propriétés ont été décrites précédemment.

Style de page

Les propriétés de style de page comprennent, outre celles du tableau 12-23, des propriétés relatives à l'en-tête et au bas de page, qui sont décrites dans une section particulière.

Tableau 12–23 Propriétés de style de page

Propriété	Type	Signification
BackColor	Long	Couleur du fond (visible avec l'aperçu).
BackTransparent	Boolean	True rend le fond transparent (interface utilisateur : sans remplissage).
TopBorder	Object	Structure de la ligne de bordure du haut. Voir cellule de tableau.
TopBorderDistance	Long	Espacement par rapport à la bordure du haut.
BottomBorder	Object	Structure de la ligne de bordure du bas. Voir cellule de tableau.
BottomBorderDistance	Long	Espacement par rapport à la bordure du bas.
LeftBorder	Object	Structure de la ligne de bordure de gauche. Voir cellule de tableau.
LeftBorderDistance	Long	Espacement par rapport à la bordure de gauche.
RightBorder	Object	Structure de la ligne de bordure de droite. Voir cellule de tableau.
RightBorderDistance	Long	Espacement par rapport à la bordure de droite.
ShadowFormat	Object	Ombre portée. Voir ombre de tableau.
TopMargin	Long	Marge du haut, en 1/100 de mm.
BottomMargin	Long	Marge du bas, en 1/100 de mm.
LeftMargin	Long	Marge de gauche, en 1/100 de mm.
RightMargin	Long	Marge de droite, en 1/100 de mm.
IsLandscape	Boolean	True si la page est en orientation Paysage.
NumberingType	Integer	Type de numérotation par défaut. Constante nommée, voir tableau 12-25. Valeur par défaut : com.sun.star.style.NumberingType.ARABIC
PageStyleLayout	Integer	Indique quelles pages sont concernées. Constante nommée, voir tableau 12-24. Valeur par défaut : com.sun.star.style.PageStyleLayout.ALL
PrinterPaperTray	String	Nom du bac de l'imprimante sélectionnée.
Width	Long	Hauteur de la page, en 1/100 de mm.
Height	Long	Largeur de la page, en 1/100 de mm.
CenterHorizontally	Boolean	True pour aligner la table horizontalement.
CenterVertically	Boolean	True pour aligner la table verticalement.
FirstPageNumber	Integer	Numéro de la première page d'impression pour cette feuille. Si la valeur vaut zéro, la numérotation suit celle de la feuille précédente.
PageScale	Integer	Echelle de réduction (en pourcentage) pour imprimer la page.
ScaleToPages	Integer	Nombre de pages pour imprimer toute la feuille.
PrintAnnotations	Boolean	True pour imprimer les notes de cellules.
PrintCharts	Boolean	True pour imprimer les diagrammes.

Tableau 12–23 Propriétés de style de page (suite)

Propriété	Type	Signification
PrintDownFirst	Boolean	True pour imprimer de haut en bas, puis les colonnes suivantes sur la droite. False pour imprimer de gauche à droite, puis les lignes suivantes en dessous.
PrintDrawing	Boolean	True pour imprimer les dessins.
PrintFormulas	Boolean	True pour imprimer le texte des formules. False pour imprimer le résultat des formules.
PrintGrid	Boolean	True pour imprimer la grille des cellules.
PrintHeaders	Boolean	True pour imprimer les en-têtes de ligne et colonnes.
PrintObjects	Boolean	True pour imprimer les objets dessin.
PrintZeroValues	Boolean	True pour imprimer les valeurs nulles.

Tableau 12–24 Constantes de disposition de page

Constante	Signification
ALL	Pages gauches et droites.
LEFT	Pages gauches seulement.
RIGHT	Pages droites seulement.
MIRRORED	Les pages gauches utilisent ce style, les pages droites prennent des valeurs "en miroir".

Tableau 12–25 Constantes de type de numérotation

Constante	Exemple de numérotation
ARABIC	1, 2, 3, 4
CHARS_UPPER_LETTER	A, B, C, D
CHARS_LOWER_LETTER	a, b, c, d
ROMAN_UPPER	I, II, III, IV, V
ROMAN_LOWER	i, ii, iii, iv, v
NUMBER_NONE	Pas de numérotation.
CHARS_UPPER_LETTER_N	A, B, ..., Y, Z, AA, BB, CC, ... AAA, ...
CHARS_LOWER_LETTER_N	a, b, ..., y, z, aa, bb, cc, ... aaa, ...

Comment utiliser les styles de page

Pour insérer dans un document des pages d'orientation différente ou ayant d'autres caractéristiques particulières, il est recommandé d'utiliser des styles de page appropriés. Tous les styles de page sont basés sur le style `Standard`. Nous allons créer deux styles :

- un style à orientation Portrait au format A4, qui est simplement une copie du style `Standard` ;
- un style à orientation Paysage, lui aussi au format A4.

La macro va créer les deux styles successivement. Dans le Styliste, basculez éventuellement l'affichage d'une famille de styles à l'autre pour faire apparaître les nouveaux styles.

```
rem Code12-07.sxc   bibli : OrientationPages Module1
Option Explicit

Sub CreerDeuxStylesPage()
Dim monDocument As Object
Dim lesFamilles As Object, uneFamille As Object
Dim nouvStyle As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("PageStyles")
nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.PageStyle")
' créer un nouveau style identique au style Standard
uneFamille.insertByName ("A4 Portrait", nouvStyle)
nouvStyle.BackColor = RGB(230,230,255)
nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.PageStyle")
' créer un style Paysage
uneFamille.insertByName ("A4 Paysage", nouvStyle)
nouvStyle.IsLandscape = True
nouvStyle.Width  = 29700 ' hauteur 29,7 cm
nouvStyle.Height = 21000 ' largeur 21,0 cm
End Sub
```

Pour chaque nouveau style, nous obtenons un objet style avec la fonction `CreateInstance` de l'objet document. Ce style est inséré dans sa famille par la méthode `insertByName` puis éventuellement modifié. Remarquez qu'avec le format Paysage, nous avons aussi changé la hauteur et la largeur, alors qu'avec l'interface utilisateur un simple clic sur le bouton Paysage suffit à intervertir ces deux valeurs.

Nous disposons maintenant de deux nouveaux styles, « A4 Portrait » et « A4 Paysage ». Nous allons les utiliser dans la macro suivante, qui va affecter le premier à une feuille, le deuxième à une autre feuille.

```
rem Code12-07.sxc   bibli : OrientationPages Module2
Option Explicit
```

```
' exécutez auparavant la routine du module 1
Sub ChangerStyleFeuille()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille3")
maFeuille.PageStyle = "A4 Portrait"
maFeuille = lesFeuilles.getByName("Feuille4")
maFeuille.PageStyle = "A4 Paysage"
End Sub
```

Sur le document modifié, vous pouvez facilement vérifier le style de page de la feuille visible : il est affiché en bas de la fenêtre Calc. Cliquez sur l'icône Aperçu et visualisez successivement les pages imprimées : vous observerez le fond bleu clair sur la zone affichée de la feuille 3 et le changement d'orientation pour la page contenant la feuille 4.

Les en-têtes et pieds de page

Les en-têtes et pieds de page ne sont pas des éléments ordinaires du document. En effet, ils sont des éléments appartenant à un style de page, par défaut le style de page `Standard`. Pour faire apparaître ou pour modifier un en-tête ou un pied de page, il faut connaître ou retrouver le style de page de la feuille qui nous intéresse, et le modifier. En conséquence, toutes les feuilles du même style subiront la même modification d'en-tête ou de pied de page.

Nous allons insérer un en-tête dans les pages d'impression de la feuille 4. Notre en-tête n'est pas différencié entre pages gauches et pages droites, il suffit de remplir celui des pages de droite.

```
rem Code12-07.sxw   bibli : Haut_Bas Module1
Option Explicit

Sub InsererUnEnTete()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, enTete As Object
Dim Texte2 As Object, Curseur2 As Object
Dim nomStyleMaPage As String, StyleMaPage As Object
Dim stylesPage As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Feuille4")
' récupérer le nom du style de page en cours
nomStyleMaPage = maFeuille.PageStyle
print "Cette page est du style : " & nomStyleMaPage
```

```
' récupérer la collection de styles de pages
stylesPage = monDocument.StyleFamilies.getByName("PageStyles")
' récupérer le style de page
StyleMaPage = stylesPage.getByName(nomStyleMaPage)
StyleMaPage.HeaderIsOn = true ' insérer un en-tête
StyleMaPage.HeaderBodyDistance = 1000 ' 10 mm
StyleMaPage.HeaderHeight = 2500 ' 25mm

' l'en-tête page droite est aussi utilisé pour la gauche
enTete = StyleMaPage.RightPageHeaderContent
Texte2 = enTete.LeftText ' zone de texte gauche de l'en-tête
Curseur2 = Texte2.createTextCursor ' curseur dans l'en-tête
' écrire un texte dans l'en-tête
Texte2.insertString(Curseur2, "Voici un en-tête", false)
StyleMaPage.RightPageHeaderContent = enTete ' mettre à jour
End Sub
```

Une fois récupéré le style de page, nous activons l'en-tête, réglons sa hauteur et sa position, et obtenons le contenu de l'en-tête. Il comporte trois zones de texte : gauche, centre, droit. Nous récupérons l'objet texte de la partie gauche. La phase d'écriture utilise toutes les possibilités d'écriture dans un texte Writer. De même, nous pourrions modifier les deux autres zones. Enfin, nous recopions le nouveau contenu dans l'en-tête du style de page.

Nous voyons encore un exemple où l'usage d'un document modèle avec des styles bien définis peut alléger considérablement l'écriture de macros en évitant de créer ou modifier les en-têtes ou bas de page.

L'objet style de page expose un très grand nombre de propriétés pour l'en-tête (`Header`) et pour le pied de page (`Footer`). Comme ces propriétés sont similaires, le tableau 12-26 ne listera que les principales propriétés de l'en-tête, sachant qu'il suffit de remplacer `Header` par `Footer` dans le nom de propriété pour obtenir celle du bas de page.

Tableau 12–26 Principales propriétés d'en-tête de page

Propriété	Type	Signification
`HeaderIsOn`	`Boolean`	`True` si un en-tête est activé.
`HeaderIsShared`	`Boolean`	`True` si les en-têtes gauche et droite sont identiques (valeur par défaut).
`HeaderBackColor`	`Long`	Couleur du fond.
`HeaderBackTransparent`	`Boolean`	`True` rend le fond transparent (interface utilisateur : sans remplissage).
`HeaderLeftBorderDistance`	`Long`	Distance entre marge et bord gauche de l'en-tête, en 1/100 de mm.
`HeaderRightBorderDistance`	`Long`	Distance entre marge et bord droit de l'en-tête, en 1/100 de mm.
`HeaderTopBorderDistance`	`Long`	Distance entre marge et bord du haut de l'en-tête, en 1/100 de mm.
`HeaderBottomBorderDistance`	`Long`	Distance entre marge et bord du bas de l'en-tête, en 1/100 de mm.
`HeaderBodyDistance`	`Long`	Distance entre en-tête et texte principal, en 1/100 de mm.

Tableau 12–26 Principales propriétés d'en-tête de page (suite)

Propriété	Type	Signification
HeaderHeight	Long	Hauteur de l'en-tête, en 1/100 de mm.
HeaderShadowFormat	Object	Ombre portée. Voir ombre de cellule.
RightPageHeaderContent	Object	Structure d'accès au texte de l'en-tête des pages droites, ou de toutes les pages, voir tableau 12-27.
LeftPageHeaderContent	Object	Structure d'accès au texte de l'en-tête des pages gauches, voir tableau 12-27.
HeaderLeftMargin	Long	Marge gauche de l'en-tête.
HeaderRightMargin	Long	Marge droite de l'en-tête.
HeaderLeftBorder	Object	Structure de la ligne de bordure de gauche. Voir bordure de cellule.
HeaderRightBorder	Object	Structure de la ligne de bordure de droite. Voir bordure de cellule.
HeaderTopBorder	Object	Structure de la ligne de bordure du haut. Voir bordure de cellule.
HeaderBottomBorder	Object	Structure de la ligne de bordure du bas. Voir bordure de cellule.

Tableau 12–27 Structure d'un contenu d'en-tête de page

Élément	Type	Signification
LeftText	Object	Zone gauche du texte de l'en-tête.
CenterText	Object	Zone centrale du texte de l'en-tête.
RightText	Object	Zone droite du texte de l'en-tête.

Configuration de Calc

Nous décrivons ici comment modifier certaines options de l'application Calc. Elles s'appliquent ensuite à chaque ouverture de document Calc. Néanmoins, vous pouvez aussi modifier ces propriétés temporairement, pendant un travail sur un document particulier.

Le service `GlobalSheetSettings` donne accès à plusieurs propriétés intéressantes, listées dans le tableau 12-28. Elles ont pour la plupart un équivalent dans le panneau du menu Outils > Options > Classeur. Elles sont accessibles en lecture et en écriture. La mise en œuvre est très simple, par exemple :

```
Dim sv As Object
sv = CreateUnoService("com.sun.star.sheet.GlobalSheetSettings")
sv.MoveSelection = false
```

Tableau 12–28 Principales propriétés de GlobalSheetSettings

Propriété	Type	Signification
MoveSelection	Boolean	True pour déplacer la sélection de cellule quand on appuie sur la touche Entrée. False pour rester sur la même cellule.
MoveDirection	Integer	Sens du déplacement du curseur de cellule à l'appui de la touche Entrée (si la propriété MoveSelection vaut True). Constante nommée, voir tableau 12-29.
UseTabCol	Boolean	Avec la valeur True, et si MoveSelection vaut True, à l'appui de la touche Entrée la sélection de cellule revient dans la colonne d'où on était parti avec une tabulation. Le déplacement est aussi combiné avec la valeur de MoveDirection.
EnterEdit	Boolean	True bascule en mode édition quand on appuie sur la touche Entrée.
ExtendFormat	Boolean	True active l'expansion du formatage.
RangeFinder	Boolean	True affiche les références en couleur.
ExpandReferences	Boolean	True active l'expansion des références.
MarkHeader	Boolean	True met en évidence la sélection dans les en-têtes de lignes et colonnes.
DoAutoComplete	Boolean	True pour activer l'AutoSaisie.
StatusBarFunction	Integer	Fonction utilisée dans la barre d'état (voir explications).
Scale	Integer	Valeur par défaut du zoom sur un nouveau document Calc, s'il n'est pas basé sur un modèle. La valeur est en pourcentage, 100 pour 100 %, ou bien : -1 pour un zoom optimal, -2 pour afficher la page entière, -3 pour s'adapter à la largeur de page.
UserLists	Array(String)	Listes de tri. Chaque liste est une chaîne de caractères dont chaque nom est séparé du suivant par une virgule.
LinkUpdateMode	Integer	Actualiser les liens au chargement : 0 : toujours, 1 : jamais, 2 : sur demande.
PrintAllSheets	Boolean	True pour imprimer toutes les feuilles. False pour n'imprimer que les feuilles sélectionnées.
PrintEmptyPages	Boolean	True pour imprimer même les feuilles vides.
UsePrinterMetrics	Boolean	True pour utiliser les paramètres de l'imprimante pour le formatage.
ReplaceCellsWarning	Boolean	True pour avertir avant d'écraser la donnée d'une cellule.

Les constantes pour `MoveDirection`, listées au tableau 12-29, sont de la forme :

```
com.sun.star.sheet.MoveDirection.RIGHT
```

Tableau 12–29 Constantes de déplacement de sélection

Constante	Signification
DOWN	Vers le bas.
UP	Vers le haut.
RIGHT	Vers la droite.
LEFT	Vers la gauche.

La propriété `StatusBarFunction` mérite quelques explications. Lorsque vous sélectionnez un groupe de cellules dans Calc, vous voyez en bas à droite dans la fenêtre Calc une indication comme : Somme=1234,56. Automatiquement, Calc a appliqué une fonction à l'ensemble des cellules de la sélection. Cette fonction est par défaut la somme des valeurs. En modifiant la propriété `StatusBarFunction`, vous pouvez choisir une autre fonction parmi celles listées au tableau 12-30. Les valeurs sont des constantes nommées de la forme :

```
com.sun.star.sheet.StatusBarFunction.AVERAGE
```

La figure 12-2 montre l'affichage d'une moyenne dans le tableur.

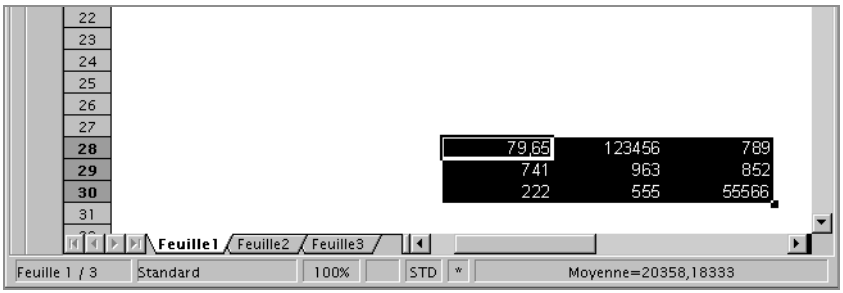


Figure 12–2
Moyenne automatique

Tableau 12–30 Constantes de fonction de barre d'état

Constante	Signification
NONE	Pas de fonction.
AVERAGE	Moyenne des valeurs des cellules à contenu numérique.
COUNTNUMS	Nombre de cellules à contenu numérique (l'affichage est : Nombre=n).
COUNT	Nombre de cellules non vides (l'affichage est : Nombre2=n).
MAX	Valeur maximale des cellules à contenu numérique.
MIN	Valeur minimale des cellules à contenu numérique.
SUM	Somme des cellules à contenu numérique.

Conclusion

Nous venons d'exposer l'utilisation de l'API pour les documents de type tableur issus de Calc. Après une description de ces objets et de leur manipulation, nous avons abordé diverses méthodes d'utilisation. Le chapitre suivant traite du maniement des objets Draw et des présentations Impress.

13

Les documents Draw et Impress

Draw est une application de dessin ; la majeure partie de ce chapitre vous montrera comment créer les figures les plus diverses grâce à des macros. Toute figure a besoin d'un support que sont les pages de dessin. Nous commencerons donc par manipuler celles-ci. Comme les concepts d'arrière-plan et de couche sont basés sur le même principe, nous les avons décrits à la suite, mais vous pourrez remettre à plus tard leur étude et sauter directement à la section « Dessiner une forme ».

Quant à Impress, c'est une application essentiellement basée sur Draw. Nous parlerons dans ce chapitre essentiellement de Draw, sachant que presque tout est applicable à Impress. Nous indiquons en fin de chapitre les éléments spécifiques à Impress.

> API **Référence sur Draw et Impress**
>
> La documentation de l'API est décrite (en anglais) dans le chapitre 9 du « Developer's Guide ». Il comporte de nombreux liens hypertextes vers les descriptions IDL du SDK.

Les pages de dessin

Par défaut, Draw présente une seule page de dessin mais il est possible d'en ajouter d'autres. Le principe est très similaire à celui des feuilles du tableur Calc, mais il existe des différences au niveau de l'interface utilisateur et au niveau de l'API.

Chaque page de dessin dans Draw comporte un onglet. Le nom par défaut des onglets est, dans la version localisée française, `Page 1`, `Page 2`, etc. Le numéro de page utilisé est le rang de l'onglet, de gauche à droite.

> **PIÈGE Faire référence à une page non renommée...**
>
> Si vous déplacez avec la souris une page non renommée, le nom sur l'onglet change pour refléter sa position, ainsi que pour les pages suivantes non renommées. Pour se référer de manière fiable à une page par son nom, il est indispensable de la renommer. Ce comportement de Draw est différent de celui de Calc.

Accéder aux pages existantes

L'objet `DrawPages` (anglais pour : pages de dessin) représente l'ensemble des pages d'un document Draw. Le nombre actuel de pages est exposé par la propriété `Count` de l'objet `DrawPages`.

Chaque page du document Draw est elle-même un objet. Basic nous permet d'obtenir facilement une de ces pages en considérant l'objet `DrawPages` comme un tableau de pages dont l'index varie de zéro au nombre de pages moins un. L'ordre des pages dans le document est reflété par la valeur de l'index. Le nom d'une page nous est indiqué dans sa propriété `Name`. En appliquant ces notions, nous allons énumérer les pages d'un document Draw :

```
rem Code13-01.sxd   bibli : Pages Module1
Option Explicit

Sub EnumererPages()
Dim monDocument As Object, lesPages As Object
Dim unePage As Object
Dim x As Long, nbP As Long

monDocument = thisComponent
lesPages = monDocument.DrawPages
nbP = lesPages.Count
print "Nombre de pages : " & nbP
for x = 0 to nbP -1
  unePage = lesPages(x)
  print "Page de rang : " & x & "  Nom : " & unePage.Name
next
End Sub
```

PIÈGE Noms visible et caché d'une page non renommée

Lisez attentivement les noms de pages affichés par la macro : elle indique pour les pages non renommées un nom différent de celui affiché sur l'onglet ! En effet, l'onglet `Page 2` est lu par la macro comme : `page2` (pas de majuscule, pas d'espace).

Deuxième piège : les noms de pages non renommées sont dans la langue de la version OpenOffice.org utilisée pour lire le document. Ainsi, *le même* document indiquera en version française : `Page 1` et dans une version anglaise : `Slide 1`.

Notez que ces comportements diffèrent totalement de la gestion des feuilles dans Calc.

Renommer une page

L'utilisateur peut changer l'ordre des pages, donc vous ne pouvez pas vous fier à l'index pour retrouver une page particulière. Le seul moyen sûr d'obtenir l'objet page que l'on souhaite consiste à utiliser son nom, et encore, à condition que la page ait été renommée. L'objet `DrawPages` nous fournit pour cela la méthode `getByName`, qui prend en argument le nom de la page souhaitée et renvoie l'objet correspondant, s'il existe. S'il n'existe pas, une erreur se produira ; aussi l'objet `DrawPages` nous donne-t-il le moyen de tester l'existence d'une page d'un nom donné avec la fonction `hasByName` qui renvoie `True` dans le cas positif.

Nous allons utiliser ces notions pour renommer une page. Pour cela, il suffit de modifier la propriété `Name` de la page. Après quoi, nous exécuterons la macro précédente pour lister les différentes pages.

```
rem Code13-01.sxd   bibli : Pages Module2
Option Explicit

Sub RenommerPage()
Dim monDocument As Object, lesPages As Object
Dim unePage As Object
Dim nom1 As String, nom2 As String
monDocument = thisComponent
lesPages = monDocument.DrawPages

nom1 = InputBox("Nom actuel de la page")
if lesPages.hasByName(nom1) then
  nom2 = InputBox("Nouveau nom pour la page")
  ' récupérer la page "nom1"
  unePage = lesPages.getByName(nom1)
  unePage.Name = nom2 ' renommer cette page
  EnumererPages  ' lister les pages du document
else
  MsgBox(nom1 & " n'existe pas",16)
end if
End Sub
```

Exécutez la macro, vous verrez l'onglet de la page concernée changer de nom.

Attention à la casse ! Les noms de pages doivent être écrits en respectant les majuscules et minuscules.

Pour le nom d'une page non encore renommée, relisez la remarque plus haut ! Le fichier du zip téléchargeable comporte deux pages non renommées, d'onglets `Page 2` et `Page 3`. Amusez-vous à renommer `Page 2` en `Page 3`...

PIÈGE Pages du même nom

La fonction API de renommage accepte sans sourciller un nom de page existante ! Vous obtenez alors deux ou plusieurs pages du même nom, et la fonction `getByName` vous renvoie une des pages existantes. L'interface utilisateur, au contraire, vérifie le nom avant de renommer la page ; faites-en autant.

En renommant une page avec une chaîne de caractères vide, la page redevient « non renommée ».

Ajouter une nouvelle page

La fonction `insertNewByIndex` de l'objet `DrawPages` sert à créer une nouvelle page vierge à droite d'une page existante (selon l'ordre des onglets). Cette méthode prend pour argument l'index de la page de référence. Comme la nouvelle page n'a pas encore été renommée, elle aura pour nom `Page n`. La valeur `n` sera égale à l'index de la page de référence, plus 2. Si la valeur de l'argument est hors limites, par exemple négative, la page sera ajoutée à la fin.

REMARQUE Insérer en première position

Il n'est pas possible d'insérer une page Draw avant la première page actuelle.

La fonction `insertNewByIndex` renvoie l'objet page nouvellement créé. Il n'est pas vraiment pratique d'insérer une page à une position, car l'utilisateur peut modifier la structure du document. Il est plus sûr de se positionner par rapport à une page dont on connaît le nom. La position d'une page Draw est exposée par sa propriété `Number`, qui vaut 1 pour la première page, alors que nous avons besoin d'un index débutant à zéro

Après avoir inséré la nouvelle page, nous la renommons ; ensuite, nous énumérons les pages du document.

```
rem Code13-01.sxd   bibli : Pages Module3
Option Explicit

Sub AjouterPage()
Dim monDocument As Object, lesPages As Object
Dim unePage As Object, P1 As Object
Dim nom1 As String, nom2 As String, idx As Long
monDocument = thisComponent
lesPages = monDocument.DrawPages
Do
  nom1 = InputBox("Insérer après la page :")
  ' reposer la question en cas d'erreur
Loop Until lesPages.hasByName(nom1)
P1 = lesPages.getByName(nom1)
idx = P1.Number -1 ' position de la page "nom1"
nom2 = InputBox("La nouvelle page aura pour nom :")
' insérer à droite de la page "nom1"
unePage = lesPages.insertNewByIndex(idx)
unePage.Name = nom2 ' renommer la page créée
EnumererPages  ' lister les pages du document
End Sub
```

Supprimer une page

La méthode `remove` de l'objet `DrawPages` supprime l'objet page fourni en argument. Cet exemple demande un nom de page à supprimer ; si la chaîne de caractères est nulle, c'est que la demande est annulée ; sinon, nous vérifions qu'une page de ce nom existe avant de la supprimer.

```
rem Code13-01.sxd   bibli : Pages Module4
Option Explicit

Sub SupprimerPage()
Dim monDocument As Object, lesPages As Object
Dim unePage As Object
Dim nom1 As String
monDocument = thisComponent
lesPages = monDocument.DrawPages

Do
  nom1 = InputBox("Page à supprimer ?")
  if nom1 = "" then exit sub ' annulation, ne rien supprimer
  ' reposer la question en cas d'erreur
Loop Until lesPages.hasByName(nom1)
unePage = lesPages.getByName(nom1) ' récupérer l'objet page
lesPages.remove(unePage) ' supprimer la page
End Sub
```

Dupliquer une page

L'objet document Draw permet de dupliquer une page existante, avec ses dessins, grâce à la fonction `duplicate`. Elle reçoit en argument l'objet page à dupliquer, et renvoie l'objet page créé. La nouvelle page sera insérée à droite de la page de référence. Ici aussi, elle sera créée « non renommée ».

```
rem Code13-01.sxd   bibli : Pages Module5
Option Explicit

Sub DupliquerPage()
Dim monDocument As Object, lesPages As Object
Dim unePage As Object, clone As Object, nom1 As String
monDocument = thisComponent
lesPages = monDocument.DrawPages
Do
  nom1 = InputBox("Page à dupliquer ?")
  if nom1 = "" then exit sub ' annulation, ne rien supprimer
  ' reposer la question en cas d'erreur
Loop Until lesPages.hasByName(nom1)
unePage = lesPages.getByName(nom1)

clone =  monDocument.duplicate(unePage)
EnumererPages  ' lister les pages du document
End Sub
```

Déplacer une page dans la liste des pages

Les fonctions de l'API ne permettent pas de modifier l'ordre des pages. Cependant, l'interface utilisateur peut le faire.

La page visible par l'utilisateur

La page visible dans l'interface utilisateur est appelée en anglais *current page* (page courante). Nous pouvons la récupérer avec la propriété `CurrentPage` de l'objet contrôleur associé au document. Pour rendre visible une autre page, il suffit de l'affecter à la propriété `CurrentPage`.

```
rem Code13-01.sxd   bibli : Pages Module6
Option Explicit

Sub PageVisible()
Dim monDocument As Object, lesPages As Object
Dim unePage As Object
Dim texte1 As String, nom2 As String
monDocument = thisComponent
lesPages = monDocument.DrawPages
```

```
unePage = monDocument.currentController.CurrentPage
texte1 = "Page active : " & unePage.Name & chr(13)
nom2 = InputBox(texte1 & "Quelle page rendre active ?")
if lesPages.hasByName(nom2) then
  unePage = lesPages.getByName(nom2)
  monDocument.currentController.CurrentPage = unePage
end if
End Sub
```

Les arrière-plans

Draw peut utiliser des pages d'arrière-plan (anglais : *Master page*). Une page ordinaire est toujours liée à une page d'arrière-plan. Tout ce qui est sur une page d'arrière-plan apparaîtra en fond pour les pages qui l'utilisent. L'interface utilisateur de Draw permet de changer l'arrière-plan d'une page en le choisissant parmi plusieurs modèles (cliquer droit sur la page, Page > Style de page). Pour accéder au mode d'affichage arrière-plan, enfoncer la minuscule icône dans le coin bas-gauche de la fenêtre.

Comme les arrière-plans sont gérés de manière un peu similaire aux pages, nous allons suivre la même démarche dans la description. Attention toutefois, il y a des différences.

Accéder aux arrière-plans existants

Nous accédons facilement à l'objet arrière-plan utilisé par une page grâce à la propriété `MasterPage` de celle-ci :

```
Dim arrPlan As Object
arrPlan = maPage.MasterPage
```

L'objet `MasterPages` (notez le « s » final) représente l'ensemble des arrière-plans d'un document Draw. Le nombre actuel d'arrière-plans est fourni par la propriété `Count` de l'objet `MasterPages`.

Chaque arrière-plan du document Draw est lui-même un objet. Basic nous permet d'obtenir facilement un des arrière-plans en considérant l'objet `MasterPages` comme un tableau d'arrière-plans dont l'index varie de zéro au nombre d'arrière-plans moins un. L'ordre des arrière-plans dans le document est reflété par la valeur de l'index. Le nom d'un arrière-plan nous est indiqué dans sa propriété `Name`.

L'énumération des arrière-plans est calquée sur celle des pages. Le document du zip téléchargeable comporte plusieurs arrière-plans pour être plus intéressant.

```
rem Code13-01.sxd   bibli : MasterP Module1
Option Explicit

Sub EnumererArrierePlans()
Dim monDocument As Object, lesArrPlans As Object
Dim unArrPlan As Object, pp As Variant
Dim x As Long, nbP As Long
monDocument = thisComponent
lesArrPlans = monDocument.MasterPages
nbP = lesArrPlans.Count
print "Nombre de pages d'arrière-plan : " & nbP
for x = 0 to nbP -1
  unArrPlan = lesArrPlans(x)
  print "Arrière-plan de rang : " & x & _
        "  Nom : " & unArrPlan.Name
next
End Sub
```

Renommer un arrière-plan

Contrairement aux pages de dessin, les méthodes `getByName` et `hasByName` n'existent pas pour les arrière-plans. Seul l'accès par index est possible, aussi utiliserons-nous la routine utilitaire de l'annexe B `getIndexByName`. Elle renvoie l'index correspondant à un nom dans une collection.

```
rem Code13-01.sxd   bibli : MasterP Module2
Option Explicit

Sub RenommerArrierePlan()
Dim monDocument As Object, lesArrPlans As Object
Dim unArrPlan As Object
Dim nom1 As String, nom2 As String, idx As Long
monDocument = thisComponent
lesArrPlans = monDocument.MasterPages

nom1 = InputBox("Nom actuel de l'arrière-plan")
idx = getIndexByName(lesArrPlans, nom1)
if idx >= 0 then
  nom2 = InputBox("Nouveau nom pour l'arrière-plan")
  ' récupérer l'arrière-plan "nom1"
  unArrPlan = lesArrPlans(idx)
  unArrPlan.Name = nom2 ' renommer cet arrière-plan
  EnumererArrierePlans  ' lister les arrière-plans du document
end if
End Sub
```

Ici, la fonction de renommage accepte un nom nul et donne effectivement un onglet avec un nom nul !

PIÈGE Arrière-plans du même nom

La fonction API de renommage accepte sans sourciller un nom d'arrière-plan existant ! Vous obtenez alors deux ou plusieurs arrière-plans du même nom. L'interface utilisateur, au contraire, vérifie le nom avant de renommer l'arrière-plan ; faites-en autant.

Ajouter un arrière-plan

La fonction `insertNewByIndex` de l'objet `MasterPages` sert à créer un nouvel arrière-plan vierge à la position d'un arrière-plan existant (selon l'ordre des onglets). Cette méthode prend pour argument l'index de la page de référence. Comme la nouvelle page n'a pas encore été renommée, elle aura pour nom `Standard`. Si la valeur de l'argument est supérieure ou égale au nombre d'arrière-plans existants, la page sera ajoutée à la fin.

VERSIONS Nom initial des arrière-plans

Dans la version française d'OpenOffice.org, un arrière-plan inséré a pour nom `Standard`. Si ce nom existe, OpenOffice.org choisit `Standard 1`, puis `Standard 2`, etc.

Dans la version anglaise d'OpenOffice.org, le nom initial est `Default`, `Default 1`, etc.

Ces noms initiaux sont gardés lorsque vous ouvrez le document dans une autre version localisée. Si vous insérez dans un même document des arrière-plans à partir de différentes localisations d'OpenOffice.org, vous obtiendrez un joli mélange de noms.

La fonction `insertNewByIndex` renvoie l'objet page nouvellement créé.

```
rem Code13-01.sxd   bibli : MasterP Module3
Option Explicit

Sub AjouterArrierePlan()
Dim monDocument As Object, lesArrPlans As Object
Dim unArrPlan As Object
Dim nom1 As String, nom2 As String, idx As Long
monDocument = thisComponent
lesArrPlans = monDocument.MasterPages

Do
  nom1 = InputBox("Insérer à l'emplacement de la page :")
  ' récupérer l'index de la page nom1
  idx = getIndexByName(lesArrPlans, nom1)
Loop Until idx>=0 ' reposer la question en cas d'erreur
nom2 = InputBox("La nouvelle page aura pour nom :")
' insérer à la place de la page "nom1", qui se décalera à droite
unArrPlan = lesArrPlans.insertNewByIndex(idx)
unArrPlan.Name = nom2 ' renommer la page créée
EnumererPagesArrierePlan  ' lister les arrière-plans du document
End Sub
```

Supprimer un arrière-plan

La méthode `remove` de l'objet `MasterPages` supprime l'objet arrière-plan fourni en argument.

```
rem Code13-01.sxd   bibli : MasterP Module4
Option Explicit

Sub SupprimerArrierePlan()
Dim monDocument As Object, lesArrPlans As Object
Dim unArrPlan As Object
Dim nom1 As String, idx As Long
monDocument = thisComponent
lesArrPlans = monDocument.MasterPages

Do
  nom1 = InputBox("Arrière-plan à supprimer ?")
  if nom1 = "" then exit sub ' annulation, ne rien supprimer
  ' reposer la question en cas d'erreur
  idx = getIndexByName(lesArrPlans, nom1)
Loop Until idx >= 0
' récupérer l'arrière-plan
unArrPlan = lesArrPlans(idx)
lesArrPlans.remove(unArrPlan) ' supprimer l'arrière-plan
End Sub
```

Il n'est pas possible de supprimer un arrière-plan utilisé par une des pages du document. La méthode `remove` ne déclenche pas d'erreur, mais ne fait rien.

Dupliquer un arrière-plan

Il n'existe pas de fonction permettant de dupliquer un arrière-plan.

Déplacer un arrière-plan dans la liste des arrière-plans

Il n'existe pas de fonction permettant de déplacer un arrière-plan dans la liste. Ce n'est pas non plus possible avec l'interface utilisateur.

Affecter un arrière-plan à une page

L'arrière-plan utilisé par une page est exposé par sa propriété `MasterPage` (notez le singulier). Pour que la page utilise un autre arrière-plan, il suffit de modifier cette propriété. La macro ci-dessous reprend des séquences déjà vues ; la seule instruction nouvelle est à la fin.

```
rem Code13-01.sxd   bibli : MasterP Module5
Option Explicit
```

```
Sub AffecterArrierePlanAPage()
Dim monDocument As Object
Dim unePage As Object, lesPages As Object
Dim unArrPlan As Object, lesArrPlans As Object
Dim idx As Long
Dim nomP As String, nomArr As String
monDocument = thisComponent

lesPages = monDocument.DrawPages
Do
  nomP = InputBox("Nom de la page ?")
  if nomP = "" then exit sub ' annulation, ne rien supprimer
  ' reposer la question en cas d'erreur
Loop Until lesPages.hasByName(nomP)
unePage = lesPages.getByName(nomP)

lesArrPlans = monDocument.MasterPages
Do
  nomArr = InputBox("Arrière-plan à appliquer ?")
  if nomArr = "" then exit sub ' annulation, ne rien supprimer
  ' reposer la question en cas d'erreur
  idx = getIndexByName(lesArrPlans, nomArr)
Loop Until idx >= 0
unArrPlan = lesArrPlans(idx)
' appliquer l'arrière-plan choisi à la page choisie
unePage.MasterPage = unArrPlan
End Sub
```

Nous avons signalé qu'il n'est pas possible de supprimer un arrière-plan utilisé par une des pages du document. Le seul moyen de vérifier qu'un arrière-plan n'est pas utilisé est de balayer toutes les pages et de vérifier l'arrière-plan que chacune utilise.

Les couches

Ici aussi, nous allons trouver des mécanismes similaires de gestion de « page », mais encore différents.

Accéder aux couches existantes

L'objet `LayerManager` (anglais pour Gestionnaire de couches) représente l'ensemble des couches d'un document Draw. Le nombre actuel de couches est fourni par la propriété `Count` de l'objet `LayerManager`. Chaque couche du document Draw est elle-même un objet. OOoBasic nous permet d'obtenir facilement une des couches en considérant l'objet `LayerManager` comme un tableau de couches dont l'index varie de zéro au nombre de couches moins un. Le nom d'une couche nous est indiqué dans sa propriété `Name`.

L'énumération des couches est calquée sur celle des pages. Le document du zip téléchargeable comporte plusieurs couches pour être plus intéressant.

```
rem Code13-01.sxd   bibli : Couches Module1
Option Explicit

Sub EnumererCouches()
Dim monDocument As Object, lesCouches As Object
Dim uneCouche As Object, pp As Variant
Dim x As Long, nbP As Long
monDocument = thisComponent

lesCouches = monDocument.LayerManager
nbP = lesCouches.Count
print "Nombre de couches : " & nbP
for x = 0 to nbP -1
  uneCouche = lesCouches(x)
  print "Couche de rang : " & x & _
        "  Nom : " & uneCouche.Name
next
End Sub
```

En exécutant la macro, vous constaterez que les couches prédéfinies affichent des noms internes qui sont en anglais :

- *layout* = Mise en page
- *controls* = Contrôles
- *measurelines* = Lignes de cote
- *background* = Arrière-plan
- *backgroundobjects* = Objets d'arrière-plan

Les couches et l'interface utilisateur

L'interface utilisateur n'affiche pas toutes les couches existantes. La couche Objets d'arrière-plan n'est visible qu'en mode Arrière-plan et mode Couche simultanés. La couche Arrière-plan n'est jamais visible. Il en résulte que l'ordre des onglets des couches ne reflète pas le tableau fourni par `LayerManager`.

Les noms des couches prédéfinies sont localisés (traduits en français) sur les onglets, mais le nom interne est en anglais. Cependant, la fonction `hasByName` reconnaît aussi le nom localisé.

> Pour information, dans la version anglaise, les onglets des couches par défaut ont aussi un nom différent du nom interne.

Renommer une couche

Comme pour les pages de dessin, le gestionnaire de couches expose les méthodes `getByName` et `hasByName`.

```
rem Code13-01.sxd   bibli : Couches Module2
Option Explicit

Sub RenommerCouche()
Dim monDocument As Object, lesCouches As Object
Dim uneCouche As Object
Dim nom1 As String, nom2 As String, idx As Long
monDocument = thisComponent
lesCouches = monDocument.LayerManager

nom1 = InputBox("Nom actuel de la couche")
if lesCouches.hasByName(nom1) then
  nom2 = InputBox("Nouveau nom pour la couche")
  uneCouche = lesCouches.getByName(nom1)
  uneCouche.Name = nom2 ' renommer cette couche
  EnumererCouches  ' lister les couches du document
end if
End Sub
```

PIÈGE Couches du même nom

La fonction API de renommage accepte sans sourciller un nom de couche existant ! Vous obtenez alors deux ou plusieurs couches du même nom. L'interface utilisateur, au contraire, vérifie le nom avant de renommer la couche ; faites-en autant.

Ici, la fonction de renommage accepte un nom nul et donne effectivement un onglet avec un nom nul !

Ajouter une couche

La fonction `insertNewByIndex` de l'objet `LayerManager` sert à créer une nouvelle couche vierge *à la position* d'une couche existante (selon l'ordre interne des couches, pas l'ordre des onglets visibles). Cette méthode prend pour argument l'index de la couche de référence. Comme la nouvelle couche n'a pas encore été renommée, elle aura pour nom `CoucheN` avec pour `N` un numéro égal au nombre total d'onglets visibles en mode « page et couches ». Si la valeur de l'argument est supérieure ou égale au nombre de couches existantes, la nouvelle sera ajoutée à la fin.

VERSIONS Nom initial des couches

Dans la version française d'OpenOffice.org, une couche insérée a pour nom `CoucheN`. Dans la version anglaise d'OpenOffice.org, le nom initial est `LayerN`. Ces noms initiaux sont gardés lorsque vous ouvrez le document dans une autre version localisée. Si vous insérez dans un même document des couches à partir de différentes localisations d'OpenOffice.org, vous obtiendrez un joli mélange de noms.

La fonction `insertNewByIndex` renvoie l'objet page nouvellement créé.

```
Rem Code13-01.sxd   bibli : Couches Module3
Option Explicit

Sub AjouterCouche()
Dim monDocument As Object, lesCouches As Object
Dim uneCouche As Object, C1 As Object
Dim nom1 As String, nom2 As String, idx As Long
monDocument = thisComponent
lesCouches = monDocument.LayerManager
Do
  nom1 = InputBox("Insérer à l'emplacement de la couche :")
  ' reposer la question en cas d'erreur
Loop Until lesCouches.hasByName(nom1)
' position de la couche "nom1"
idx = getIndexByName(lesCouches, nom1)
nom2 = InputBox("La nouvelle couche aura pour nom :")
' insérer à la position de la couche "nom1"
uneCouche = lesCouches.insertNewByIndex(idx)
uneCouche.Name = nom2 ' renommer la couche créée
EnumererCouches  ' lister les couches du document
End Sub
```

Supprimer une couche

La méthode `remove`, de l'objet `LayerManager`, supprime l'objet couche fourni en argument.

```
rem Code13-01.sxd   bibli : Couches Module4
Option Explicit

Sub SupprimerCouche()
Dim monDocument As Object, lesCouches As Object
Dim uneCouche As Object
Dim nom1 As String
monDocument = thisComponent
lesCouches = monDocument.LayerManager

Do
  nom1 = InputBox("Couche à supprimer ?")
  if nom1 = "" then exit sub ' annulation, ne rien supprimer
```

```
  ' reposer la question en cas d'erreur
Loop Until lesCouches.hasByName(nom1)
uneCouche = lesCouches.getByName(nom1)
lesCouches.remove(uneCouche)
End Sub
```

Il est possible de supprimer une couche utilisée par des formes (dessins) du document. Celles-ci disparaissent avec la couche.

> **PIÈGE Couches par défaut**
>
> Il est possible par programmation de supprimer les couches par défaut. OpenOffice.org manifeste ensuite sa désapprobation avec une erreur logicielle et un plantage à la fermeture du fichier ! Donc, ne faites pas ça.

Dupliquer une couche

Il n'existe pas de fonction permettant de dupliquer une couche.

Déplacer une couche dans la liste des couches

Il n'existe pas de fonction permettant de déplacer une couche dans la liste. Ce n'est pas non plus possible avec l'interface utilisateur.

Les propriétés d'une couche

En dehors de la propriété `Name`, de type `String`, que nous avons déjà utilisée, une couche dispose de trois autres propriétés intéressantes, toutes de type `Boolean`, qu'on peut lire et modifier :

- `IsLocked` : si `True`, la couche est verrouillée.
- `IsPrintable` : si `True`, les objets de la couche sont imprimables.
- `IsVisible` : si `True`, les objets de la couche sont visibles.

Changement du mode d'affichage

Le mode d'affichage (mode `Page`, mode `Arrière-plan`, mode `Couche`) dépend de deux indicateurs booléens de l'objet `currentController` associé au document :

- `isMasterPageMode` doit être à la valeur `True` pour afficher le mode `Arrière-plan`, à `False` pour afficher le mode `Page`.
- `IsLayerMode` doit être à la valeur `True` pour afficher les couches de page ou d'arrière-plan.

Ces indicateurs peuvent aussi être lus.

> Il devrait en principe être possible de changer l'arrière-plan visible en modifiant la propriété `CurrentPage` de l'objet `currentController`. En réalité, l'effet est imprévisible en raison de différents bogues.

Les propriétés d'une page de dessin

Le tableau 13-1 liste les principales propriétés d'une page de dessin. Les arrière-plans ont les mêmes propriétés, excepté bien sûr `MasterPage` qui n'existe pas.

Tableau 13–1 Propriétés d'une page de dessin

Propriété	Type	Signification
`Name`	`String`	Le nom de la page.
`BorderLeft`	`Long`	Marge de gauche en 1/100 de mm.
`BorderRight`	`Long`	Marge de droite en 1/100 de mm.
`BorderTop`	`Long`	Marge du haut en 1/100 de mm.
`BorderBottom`	`Long`	Marge du bas en 1/100 de mm.
`Height`	`Long`	Hauteur totale de la page en 1/100 de mm.
`Width`	`Long`	Largeur totale de la page en 1/100 de mm.
`Number`	`Integer`	Rang de la page dans l'ordre des onglets, numérotation à partir de 1 ; en lecture seulement.
`Orientation`	`Long`	Orientation de la page à l'impression, constante nommée.
`MasterPage`	`Object`	Arrière-plan utilisé par la page.
`Count`	`Long`	Nombre de formes sur la page.
`Forms`	`Object`	Collection des formes sur la page.

La propriété `Orientation` ne concerne que l'impression, elle ne change pas la visualisation de la page. Elle ne peut prendre que deux valeurs :

```
com.sun.star.view.PaperOrientation.PORTRAIT
com.sun.star.view.PaperOrientation.LANDSCAPE
```

Attention à la casse ! Les constantes nommées doivent être écrites en respectant les majuscules et minuscules.

Dessiner une forme

Nous appellerons forme (anglais *shape*) un dessin élémentaire, par exemple un rectangle ou une ellipse. Ne confondez pas avec l'anglais *form*, qui désigne un formulaire dans le jargon OpenOffice.org.

Les formes sont dessinées sur une page de dessin, c'est-à dire une page ordinaire de Draw ou une diapo d'Impress, ou un arrière-plan.

Dessiner une forme consiste à ajouter une nouvelle forme sur une page de dessin ou sur un arrière-plan. Voici un exemple, les commentaires suivent.

```
rem Code13-02.sxd   bibli : Ajouter Module1
Option Explicit

Sub AjouterEllipse()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim dimensionsForme As New com.sun.star.awt.Size
Dim positionForme As New com.sun.star.awt.Point
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")

maForme = monDocument.createInstance(_
          "com.sun.star.drawing.EllipseShape")
dimensionsForme.Width = 13400 ' 134 mm de large
dimensionsForme.Height = 2530 ' 25,3 mm de haut
positionForme.x = 2500 ' 25 mm à droite du coin de la page
positionForme.y = 5300 ' 53 mm en dessous du coin de la page

maForme.Size = dimensionsForme
maForme.Position = positionForme
maPage.add(maForme)
End Sub
```

Dans un premier temps, nous obtenons une forme virtuelle grâce à la fonction `createInstance` de l'objet document. Nous avons choisi une ellipse, d'autres variantes de formes existent. Avant de l'ajouter sur la page, il faut la dimensionner et préciser où elle sera positionnée.

Les dimensions d'une forme sont celles du rectangle qui l'englobe. Pour les indiquer, nous avons besoin d'une structure `com.sun.star.awt.Size` (Dimension se traduit en anglais par *Size*). Elle comporte deux éléments :

- `Width` : la largeur de la forme, en 1/100 de mm,
- `Height` : la hauteur de la forme, en 1/100 de mm.

La position d'une forme est définie par les distances horizontale et verticale entre le coin haut-gauche de la page de dessin et le coin haut-gauche du rectangle qui englobe la

forme. Pour indiquer une position, nous utilisons une structure `com.sun.star.awt.Point`, qui comporte deux éléments :

- `x` : la distance horizontale entre le point de référence et le coin de la page, en 1/100 de mm,
- `y` : la distance verticale entre le point de référence et le coin de la page, en 1/100 de mm.

> **À RETENIR Position dans l'interface utilisateur**
>
> Les coordonnées de position affichées par l'interface utilisateur se réfèrent au coin haut-gauche de la zone imprimable de la page. Il y a donc une différence, correspondant aux marges horizontale et verticale.

Une fois ces deux structures remplies, nous les copions sur les propriétés `Size` et `Position` de l'objet forme. Avant de définir d'autres propriétés, que nous décrirons plus loin, il faut ajouter notre forme virtuelle sur la page grâce à sa méthode `add`. La forme apparaît. Il est encore possible de modifier les propriétés de position et dimensions de la forme après l'avoir ajoutée. Pour ajouter une forme à un arrière-plan, la seule différence est qu'il faut utiliser la méthode `add` de l'arrière-plan au lieu de celle de la page de dessin.

Les différents types de forme (rectangle, ellipse, etc.) sont décrits plus loin. Auparavant, nous allons voir comment retrouver une forme existante.

Trouver une forme existante

Trouver une forme nommée

À partir de la version 1.1 d'OpenOffice.org, l'utilisateur peut nommer une forme (faire un clic droit sur la forme sélectionnée). En fait, il peut nommer ainsi tout objet sur la page. Le nom de la forme est exposé dans sa propriété `Name`, de type `String`. Par programmation, nommer une forme se fait très simplement :

```
maForme.Name = "Ma première forme"
```

> **PIÈGE Noms des objets sur une page**
>
> Avec l'interface utilisateur, il est impossible de donner le même nom à deux objets (dessin, image, etc.) sur une même page. À l'inverse, l'API n'effectue aucun contrôle quand vous modifiez la propriété `Name` !

Malheureusement, l'API ne sait pas retrouver une forme par son nom, mais seulement par un index. Les formes (et les images) d'une page de dessin sont accessibles en Basic par simple indexation de l'objet page, par exemple :

```
maForme = maPage(3)
```

Ceci ne nous avance pas beaucoup, car nous ne connaissons pas l'ordre des dessins dans l'objet page. Nous allons utiliser une fonction utilitaire de l'annexe B. La fonction `FindObjectByName` prend comme argument l'objet page de dessin et le nom du dessin recherché. Elle explore la collection des formes et images et retourne l'objet qui a le nom demandé. En cas d'échec, la fonction renvoie un objet `Null`.

Un exemple typique, que nous retrouverons souvent dans la section consacrée aux propriétés des formes, est reproduit ci-dessous. Nous obtenons directement la page nommée `Dessin3`, puis nous utilisons `FindObjectByName` pour obtenir l'objet forme nommé `F4` qui se trouve dans cette page.

```
Dim monDocument As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin3")
maForme = FindObjectByName(maPage, "F4")
```

Trouver les formes sélectionnées par l'utilisateur

S'il y a sélection, les éléments sélectionnés sont obligatoirement sur la page de dessin visible. Or, nous savons déjà retrouver la page visible. L'objet `CurrentSelection`, exposé par le document Draw ou Impress, nous fournit la collection des formes sélectionnées par l'utilisateur. Attention toutefois, s'il n'a rien sélectionné, cet objet est `Null`. Chaque objet de la collection est accessible par simple indexation de la collection.

Ces remarques sont mises en œuvre dans le code suivant, qui affiche le texte de chacune des formes sélectionnées.

```
rem Code13-04.sxd   bibli : Selections Module1
Option Explicit

Sub SelectionsUtilisateur()
Dim monDocument As Object, maPage As Object
Dim lesFormes As Object, maForme As Object, n As Long
monDocument = thisComponent
maPage = monDocument.CurrentController.CurrentPage
lesFormes = monDocument.CurrentSelection
if IsNull(lesFormes) then
  print "Aucune sélection !"
```

```
else
  for n = 0 to lesFormes.Count -1
    maForme = lesFormes(n)
    print n, maForme.Text.String
  next
end if
End Sub
```

Vous remarquerez en faisant des essais que l'ordre des formes dans la collection n'est pas toujours identique à l'ordre des sélections.

Sélectionner visiblement une forme

Supposons que nous ayons besoin de sélectionner une forme nommée `T01` qui se trouve dans la page nommée `Dessins` du document. En sélectionnant la forme avec la méthode `Select` de l'objet `CurrentController`, la page est automatiquement rendue visible.

```
rem Code13-04.sxd   bibli : Selections Module2
Option Explicit

Sub SelectionnerForme()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessins")
maForme = FindObjectByName(maPage, "T01")
if IsNull(maForme) then
  print "Il n'existe aucune forme de ce nom"
else
  monDocument.CurrentController.Select(maForme)
end if
End Sub
```

Lister les formes d'une page

L'ensemble des formes (et images) d'une page est disponible dans une collection gérée par la page. La propriété `Count` de l'objet page nous renseigne sur le nombre d'objets sur la page et, grâce à Basic, nous avons accès à chacun d'eux par simple indexation de l'objet page.

Dans cet exemple, nous allons énumérer les formes ordinaires d'une page. Nous éviterons de lister les autres objets éventuels : image, diagramme, dessin 3D, en testant si le service `LineProperties` est reconnu. Nous afficherons le type de chaque forme en utilisant la propriété `ShapeType`.

```
rem Code13-04.sxd   bibli : Selections Module6
Option Explicit

Sub EnumererFormes2D()
Dim monDocument As Object, maPage As Object
Dim lesFormes As Object, maForme As Object
Dim nomP As String, n As Long
' rechercher seulement les formes 2D
Const serv = "com.sun.star.drawing.LineProperties"
monDocument = thisComponent
nomP = InputBox("Donnez un nom de page")
if not monDocument.DrawPages.hasByName(nomP) then exit sub
maPage = monDocument.DrawPages.getByName(nomP)
for n = 0 to maPage.Count -1
  maForme = maPage(n)
  if maForme.supportsService(serv) then
    monDocument.CurrentController.Select(maForme)
    MsgBox("Forme " & n  & chr(13) & "Type = " & _
       maForme.ShapeType & chr(13) & " Nom = " & maForme.Name)
  end if
next
monDocument.CurrentController.Select(Null) ' plus de sélection
End Sub
```

Supprimer une forme

La méthode `remove` de l'objet page de dessin supprime la forme fournie en argument.

```
rem Code13-02.sxd   bibli : Ajouter Module2
Option Explicit

Sub SupprimerForme()
Dim monDocument As Object, maPage As Object
Dim lesFormes As Object, maForme As Object
monDocument = thisComponent
lesFormes = monDocument.DrawPages
maPage = lesFormes.getByName("Dessin2")
maForme = FindObjectByName(maPage, "cube1")
maPage.remove(maForme)
End Sub
```

Le principe est identique pour supprimer une forme d'une couche ou d'un arrière-plan.

Propriétés des formes

Pour manipuler les propriétés d'une forme, nous utiliserons des formes nommées dans une page de dessin du document `Code13-02.sxd`.

Type de la forme

La propriété `ShapeType`, de type `String`, indique le type de la forme. Exemple :

```
"com.sun.star.drawing.EllipseShape"
```

La liste des différents types de formes possibles est indiquée au tableau 13-18 de la section « Les différentes formes ».

Position et taille de la forme

La position d'une forme est définie par sa propriété `Position`, tandis que sa taille est définie par sa propriété `Size`. Nous avons détaillé ces propriétés à la section « Dessiner une forme ». Vous pouvez naturellement modifier ces valeurs pour toute forme existante.

Une forme possède deux propriétés de type `Boolean` qui interdisent à l'utilisateur de modifier la taille ou la position si vous leur affectez la valeur `True` :

- `SizeProtect` interdit de modifier la taille.
- `MoveProtect` interdit de déplacer la forme.

Néanmoins, par programmation vous pouvez toujours modifier taille et position.

Les liens entre forme et couche

Toute forme est associée à une couche, et une seule. Si vous n'avez pas défini de couche particulière, une forme d'une page de dessin est associée à la couche `layout` (c'est l'onglet Mise en page, en mode Couche). La forme expose deux propriétés :

- `LayerName`, de type `String`, est le nom interne de la couche associée.
- `LayerID`, de type `Integer`, est le rang interne de la couche associée.

Le gestionnaire de couches (aussi appelé `LayerManager`), indique dans quelle couche se trouve une forme avec la fonction `getLayerForShape`. Cette dernière reçoit en argument un objet forme et renvoie un objet couche.

L'exemple qui suit liste, pour chaque forme de la page de dessin, le nom de la forme et le nom de la couche associée, obtenu des deux manières.

```
rem Code13-02.sxd   bibli : CoucheForme Module1
Option Explicit
```

```
Sub AfficherCoucheDesObjets()
Dim monDocument As Object, x As Long
Dim maPage As Object, maForme As Object
Dim lesCouches As Object, maCouche As Object
monDocument = thisComponent
lesCouches = monDocument.LayerManager
maPage = monDocument.DrawPages.getByName("Dessin4")
for x = 0 to maPage.Count -1 ' toutes les formes de la page
  maForme = maPage(x)
  maCouche = lesCouches.getLayerForShape(maForme)
  ' deux manières de trouver le nom de la couche
  print maForme.Name, maCouche.Name, maForme.LayerName
next
End Sub
```

Pour associer une forme à une autre couche, l'API nous offre encore deux moyens :

- changer le nom de la couche dans la propriété `LayerName` de la forme,
- utiliser la méthode `attachShapeToLayer` du gestionnaire de couches.

Nous allons utiliser la première manière pour modifier l'affectation des formes sélectionnées par l'utilisateur. Exceptée la ligne concernant l'association, la macro utilise des principes décrits dans les sections précédentes.

```
rem Code13-02.sxd   bibli : CoucheForme Module2
Option Explicit

Sub AssocierFormeAcouche()
Dim monDocument As Object, f As Long, nf As String
Dim lesFormes As Object, maForme As Object
Dim lesCouches As Object, maCouche As Object
monDocument = thisComponent
lesCouches = monDocument.LayerManager
Do
  nf = InputBox("Nom de la couche à associer" & chr(13) & _
               " aux formes sélectionnées")
Loop Until lesCouches.hasByName(nf)
lesFormes = monDocument.CurrentSelection
if IsNull(lesFormes) then exit sub
for f = 0 to lesFormes.Count -1 ' les formes sélectionnées
  maForme = lesFormes(f)
  maForme.LayerName = nf ' associer à la couche choisie
next
End Sub
```

La deuxième manière se ferait ainsi :

```
lesCouches.attachShapeToLayer(maForme, lesCouches.getByName(nf))
```

Le contour de la forme

Le contour d'une forme utilise les propriétés de ligne, dont le tableau 13-2 liste les principales. Certaines de ces propriétés sont expliquées davantage dans le texte. Si vous souhaitez étudier dans le détail les possibilités de ligne, consultez dans l'API la page sur le service : `com.sun.star.drawing.LineProperties`.

Tableau 13–2 Propriétés de ligne

Propriété	Type	Signification
`LineStyle`	`Long`	Genre de la ligne ; constante nommée.
`LineColor`	`Long`	Couleur de la ligne.
`LineWidth`	`Long`	Largeur de la ligne, en 1/100 de mm.
`LineTransparence`	`Integer`	Pourcentage de transparence, entre 0 (opaque) et 100 (transparent).
`LineDashName`	`String`	Nom d'un tireté prédéfini.
`LineDash`	`Object`	Structure décrivant le tireté.
`LineJoint`	`Long`	Manière de relier deux lignes ; constante nommée.

Le genre de la ligne, `LineStyle`, peut prendre les valeurs du tableau 13-3. Ce sont des constantes nommées de la forme :

```
com.sun.star.drawing.LineStyle.SOLID
```

Tableau 13–3 Constantes de LineStyle

Constante	Signification
`NONE`	Pas de ligne, donc pas de contour apparent.
`SOLID`	Ligne continue (valeur par défaut).
`DASH`	Ligne tiretée.

Les lignes tiretées

Les difficultés apparaissent avec les lignes tiretées. Le plus simple pour indiquer quel tireté utiliser est de remplir la propriété `LineDashName` avec le nom d'un tireté existant. Si vous utilisez le nom français d'un tireté prédéfini, vous risquez des problèmes si le document est affiché par une version non française d'OpenOffice.org. Même si vous employez le nom anglais, vous êtes tributaire d'une version particulière. Il vaut mieux définir dans un document modèle les tiretés dont vous aurez besoin, et leur donner un nom. C'est ce qui a été fait dans le document du zip téléchargeable contenant l'exemple suivant.

```
rem Code13-02.sxd   bibli : Contour Module1
Option Explicit

Sub AppliquerTirete()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin3")
maForme = FindObjectByName(maPage, "F4")

With maForme
  .LineDashName = "TiretPerso01"
  .LineStyle = com.sun.star.drawing.LineStyle.DASH
  .LineColor = RGB(100,0,75)
  .LineWidth = 450 ' 4,5 mm
  .LineTransparence = 50 ' semi-transparent
End With
End Sub
```

L'instruction `With` évite de répéter `maForme` pour chacune des lignes suivantes. Le tireté n'apparaît que lorsque `LineStyle` reçoit la valeur appropriée. Nous avons profité de l'exemple pour utiliser d'autres propriétés de ligne.

La propriété `LineDash` décrit précisément le tiret. C'est une structure décrite au tableau 13-4.

Tableau 13–4 Éléments de définition d'un tireté de ligne

Élément	Type	Signification
`Style`	`Long`	Genre du tiret ; constante nommée.
`Dots`	`Integer`	Nombre de points dans le tireté.
`DotLen`	`Long`	Longueur d'un point, en 1/100 de mm.
`Dashes`	`Integer`	Nombre de tirets dans le tireté.
`DashLen`	`Long`	Longueur d'un tiret, en 1/100 de mm.
`Distance`	`Long`	Distance entre les points ou les tirets, en 1/100 de mm.

Le genre du tiret (élément `Style` du tireté) est une constante nommée dont les valeurs possibles sont listées dans le tableau 13-5. Exemple de valeur :

```
com.sun.star.drawing.DashStyle.RECT
```

Pour imposer par programmation la forme du tireté, il est nécessaire de remplir un objet ayant la structure `LineDash`, puis de l'appliquer à la propriété `LineStyle` de la forme.

```
rem Code13-02.sxd   bibli : Contour Module1
Option Explicit

Sub DefinirTirete()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim mesTirets As New com.sun.star.drawing.LineDash
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin3")
maForme = FindObjectByName(maPage, "F4")

With mesTirets
  .Style = com.sun.star.drawing.DashStyle.RECT
  .Dots = 4       ' 4 points
  .DotLen = 50    ' de 0,5 mm
  .Dashes = 2     ' suivis de 2 tirets
  .DashLen = 200  ' de 2 mm
  .Distance = 150 ' espacés de 1,5 mm
End With
maForme.LineWidth = 100 ' 1 mm d'épaisseur
maForme.LineDash = mesTirets
maForme.LineStyle = com.sun.star.drawing.LineStyle.DASH
End Sub
```

Tableau 13–5 Constantes de DashStyle

Constante	Signification
RECT	En forme de rectangle.
ROUND	En forme de disque.
RECTRELATIVE	En forme de rectangle proportionnel à la longueur de la ligne.
ROUNDRELATIVE	En forme de disque proportionnel à la longueur de la ligne.

Le fond et la forme

Sans instruction particulière, la forme est créée avec une couleur de fond uniforme, dont le coloris est une valeur par défaut. Toutefois, d'autres styles de remplissage du fond sont possibles, selon la valeur de la propriété `FillStyle`.

Les styles de remplissage du fond

Le terme « style » employé ici a seulement le sens de variante. La propriété `FillStyle` d'une forme est de type `Long`, et contient une constante nommée (tableau 13-6), de la forme :

```
com.sun.star.drawing.FillStyle.SOLID
```

Tableau 13–6 Constantes de style de remplissage

Constante	Signification
NONE	Fond invisible (quelle que soit la valeur de FillTransparence).
SOLID	Couleur uniforme (valeur par défaut).
GRADIENT	Dégradé de couleur.
HATCH	Hachuré.
BITMAP	Utilisation d'un motif bitmap (une image répétée).

Nous allons expliquer comment utiliser chacun de ces styles de remplissage. Si vous souhaitez étudier dans le détail les possibilités de dégradé, quadrillage, etc, consultez l'API à la page sur le service : com.sun.star.drawing.FillProperties

Couleur de fond

Pour imposer une couleur, on donne à la propriété FillColor de la forme une valeur de couleur.

```
rem Code13-02.sxd   bibli : Proprietes Module1
Option Explicit

Sub CouleurDuFondForme()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")

maForme.FillColor = RGB(100,255,255)
End Sub
```

La couleur obtenue est répartie uniformément sur la surface de la forme. OpenOffice.org offre une alternative, le dégradé de couleur.

Fond en dégradé de couleur

Créer par l'API un dégradé de couleurs est assez complexe, et le résultat ne peut être jugé que visuellement. Alors, il est préférable d'utiliser un modèle de document dans lequel on aura défini des dégradés personnels avec l'interface utilisateur.

Pour visualiser les dégradés prédéfinis ou ajouter des dégradés dans un document, sélectionnez une forme, cliquez droit, choisissez Remplissage, onglet Dégradé. Vous aurez ainsi une idée de la quantité de paramètres utilisables et de leurs effets. Chaque dégradé défini comporte un nom. Il nous suffira d'appliquer ce nom à la propriété FillGradientName.

Auparavant, nous devons faire mémoriser à notre document (final, ou le modèle) les dégradés que nous souhaitons utiliser. Il suffit de les appliquer successivement à une forme quelconque (Remplissage, onglet Remplissage), puis sauver le document.

CONSEIL

Pour éviter des incompatibilités entre versions ou localisation d'OpenOffice.org, il est recommandé de n'utiliser en programmation que des dégradés personnels auxquels on aura donné un nom.

Pour accéder aux dégradés disponibles dans un document, nous devons créer un objet service `GradientTable` à partir du document. On obtient un conteneur d'objets nommés, qui sont les dégradés. Pour savoir si un dégradé est présent dans le document, nous utiliserons la fonction `hasByName`. La fonction `getByName` renverra le dégradé correspondant au nom.

```
rem Code13-02.sxd   bibli : Degrades Module2
Option Explicit

Sub ObtenirDegrade()
Dim monDocument As Object
Dim servGrad As Object, nomDegrade As String, unGrad As Object
monDocument = thisComponent
servGrad = monDocument.createInstance(_
           "com.sun.star.drawing.GradientTable")

nomDegrade = "Radial BleuTurquoise horizontal"
if servGrad.hasByName(nomDegrade) then
  unGrad = servGrad.getByName(nomDegrade)
  print "Centre X : " & unGrad.XOffset & "%"
  print "Centre Y : " & unGrad.YOffset & "%"
end if
End Sub
```

Attention à la casse ! Les noms de dégradés doivent être écrits en respectant les majuscules, les minuscules, et les espaces.

Un dégradé est une structure dont les éléments sont indiqués dans le tableau 13-7. Nous n'irons pas plus loin dans leur description.

Maintenant, nous en savons assez pour affecter un dégradé à une forme : récupérer la forme, vérifier que le dégradé existe bien, appliquer le nom du dégradé sur la propriété `FillGradientName` (de type `String`) de la forme, et ne pas oublier de changer la propriété `FillStyle` de la forme pour indiquer que le remplissage est un dégradé.

```
rem Code13-02.sxd   bibli : Degrades Module3
Option Explicit

Sub AppliquerDegrade()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim servGrad As Object, nomDegrade As String, unGrad As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")
nomDegrade = "Radial BleuTurquoise horizontal"
servGrad = monDocument.createInstance(_
           "com.sun.star.drawing.GradientTable")
' sage précaution : vérifier que le nom existe !
if servGrad.hasByName(nomDegrade) then
  maForme.FillGradientName = nomDegrade
  maForme.FillStyle = com.sun.star.drawing.FillStyle.GRADIENT
else
  MsgBox("Nom de dégradé inconnu : " & nomDegrade, 16)
end if
End Sub
```

Pour un dégradé prédéfini, nous pouvons utiliser aussi bien le nom anglais que le nom francisé. Notez que l'objet dégradé de couleur employé par une forme est disponible dans sa propriété `FillGradient`.

Tableau 13–7 Propriétés d'un dégradé

Propriété	Type	Signification
`Style`	`Long`	Genre du dégradé (linéaire, axial, radial, etc.) ; constante nommée.
`StartColor`	`Long`	Couleur initiale.
`EndColor`	`Long`	Couleur finale.
`Angle`	`Integer`	Angle, en 1/10 de degré, positif dans le sens inverse des aiguilles d'une montre.
`Border`	`Integer`	Pourcentage de la largeur totale où la couleur initiale est utilisée ; valeur de 0 à 100.
`XOffset`	`Integer`	Pourcentage de la largeur totale où le dégradé commence ; valeur de 0 à 100.
`YOffset`	`Integer`	Pourcentage de la hauteur totale où le dégradé commence ; valeur de 0 à 100.
`StartIntensity`	`Integer`	Intensité au début du dégradé.
`EndIntensity`	`Integer`	Intensité à la fin du dégradé.
`StepCount`	`Integer`	Nombre de pas de changement de couleur ; zéro si ce nombre n'est pas limité.

Un peu de transparence

La couleur, uniforme ou en dégradé, est opaque par défaut, mais on peut régler sa transparence par la propriété `FillTransparence`, de type `Integer`. Elle prend une valeur entre 0 et 100 qui est le pourcentage de transparence ; 0 pour une couleur opaque, 100 pour une couleur totalement transparente.

```
rem Code13-02.sxd   bibli : Proprietes Module2
' - - - extrait spécifique à l'exemple - - -

maForme.FillTransparence = 65
```

Cette transparence est uniforme sur toute la surface de la forme. Il est toutefois possible d'obtenir un dégradé de la transparence.

Le dégradé de transparence

Le principe du dégradé de transparence est similaire à ce qui a été exposé pour un dégradé de couleur. Les différences sont les suivantes :

- Le service de dégradé a pour nom `com.sun.star.drawing.TransparencyGradientTable`.
- La propriété `FillStyle` ne le concerne pas.
- La propriété de nom de dégradé de transparence s'intitule `FillTransparenceGradientName`.
- La propriété contenant le dégradé de transparence d'une forme a pour nom `FillTransparenceGradient`.
- L'utilisateur n'a pas accès au nom du dégradé de transparence.

Le dernier point demande de plus amples explications. Quand l'utilisateur remplit une forme avec un dégradé de transparence, le système crée un dégradé nommé `Transparency n`. Le nombre `n` est choisi dynamiquement. Cependant, comme l'utilisateur ne peut donner un nom à son dégradé, celui-ci est supprimé du document quand il n'est plus utilisé.

Pour contourner cette limitation de l'interface utilisateur, nous utiliserons une macro qui sert à nommer des dégradés de transparence dans un document. Le principe consiste à utiliser une forme quelconque du document, la remplir avec un dégradé de transparence, exécuter la macro qui lui donnera un nom, mettre un autre dégradé de transparence sur la même forme, exécuter de nouveau la macro, etc. Nous avons procédé ainsi sur le document du zip téléchargeable pour créer les dégradés :

- `Lin 0 0 0 100`
- `Radial 50 50 0 0 100`
- `Radial 50 50 0 100 0`

Les noms correspondent aux paramètres utilisés.

```
rem Code13-02.sxd   bibli : Transparence Module1
Option Explicit

Sub NommerDegradeTransparent()
Dim monDocument As Object, maPage As Object
Dim maForme As Object, nouvNom As String
Dim servGrad As Object, nomDegrade As String, unGrad As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")

nomDegrade = maForme.FillTransparenceGradientName
servGrad = monDocument.createInstance(_
           "com.sun.star.drawing.TransparencyGradientTable")
if servGrad.hasByName(nomDegrade) then
  unGrad = servGrad.getByName(nomDegrade)
  nouvNom = InputBox("Nom du nouveau dégradé transparent")
  if nouvNom <> "" then servGrad.insertByName(nouvNom, unGrad)
else
  MsgBox("Nom de dégradé inconnu : " & nomDegrade, 16)
end if
End Sub
```

Nous récupérons le nom du dégradé créé automatiquement pour la forme, puis nous obtenons l'objet dégradé ayant ce nom. La méthode `insertByName` de l'objet collection de dégradés ajoute un nouveau dégradé ayant le nom donné par l'utilisateur. Ce dégradé restera mémorisé alors que le premier disparaîtra.

EXERCICE

Nous laisserons le lecteur créer une macro pour supprimer de la collection un dégradé de transparence, avec la méthode `removeByName` qui prend en argument le nom de l'objet à supprimer.

Appliquer un dégradé de transparence est quasi identique à ce que nous avons déjà vu.

```
rem Code13-02.sxd   bibli : Transparence Module3
Option Explicit

Sub AppliquerDegradeTransparent()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim servGrad As Object, nomDegrade As String, unGrad As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")
nomDegrade = "Radial 50 50 0 100 0"
```

```
servGrad = monDocument.createInstance(_
           "com.sun.star.drawing.TransparencyGradientTable")
' sage précaution : vérifier que le nom existe !
if servGrad.hasByName(nomDegrade) then
  maForme.FillTransparenceGradientName = nomDegrade
else
  MsgBox("Nom de dégradé inconnu : " & nomDegrade, 16)
end if
End Sub
```

Il est parfaitement possible de combiner sur une même forme un dégradé de couleur et un dégradé de transparence.

Fond hachuré

Nous retrouvons ici des notions similaires à celles développées pour les dégradés de couleur, aussi la description sera-t-elle plus succincte.

Hachurer se traduit en anglais par *to hatch* (à vos souhaits !). Comme pour un dégradé de couleur, une forme dispose de la propriété `FillHatchName`, de type `String`, qui indique le nom du modèle de hachures, et de la propriété `FillHatch` qui contient l'objet hachure, qui est une structure décrite dans le tableau 13-8.

Tableau 13–8 Propriétés de hachure

Propriété	Type	Signification
`Style`	`Long`	Style de hachure ; constante nommée.
`Color`	`Long`	Couleur des lignes.
`Distance`	`Long`	Écartement des lignes, en 1/100 de mm.
`Angle`	`Long`	Angle de rotation des lignes, en 1/10 de degré, positif dans le sens inverse des aiguilles d'une montre.

Les styles de hachures sont des constantes nommées (tableau 13-9) de la forme :

```
com.sun.star.drawing.HatchStyle.SINGLE
```

Tableau 13–9 Constantes de style de hachure

Constante	Élément de base de la hachure
`SINGLE`	Une ligne horizontale.
`DOUBLE`	Une ligne horizontale et une ligne verticale.
`TRIPLE`	Une ligne horizontale, une ligne verticale, une ligne en diagonale.

L'interface utilisateur permet de définir et de nommer des styles de hachure. Il est préférable de définir dans un document modèle quelques styles personnels, plutôt que d'utiliser des styles prédéfinis en standard. Ici, nous emploierons le service `com.sun.star.drawing.HatchTable` pour obtenir une hachure dont on connaît le nom. Le style de hachure est appliqué à une forme en copiant son nom dans la propriété `FillHatchName` et en affectant à la propriété `FillStyle` la constante nommée `HATCH`.

```
rem Code13-02.sxd   bibli : Hachures Module2
Option Explicit

Sub AppliquerHachure()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim servHach As Object, nomHach As String
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")
nomHach = "VertClair croisé 2mm 30dg"
servHach = monDocument.createInstance(_
           "com.sun.star.drawing.HatchTable")
' sage précaution : vérifier que le nom existe !
if servHach.hasByName(nomHach) then
  maForme.FillHatchName = nomHach
  maForme.FillStyle = com.sun.star.drawing.FillStyle.HATCH
else
  MsgBox("Nom de hachure inconnu : " & nomHach, 16)
end if
End Sub
```

Les hachures ont un fond transparent si la propriété `FillBackground`, de type `Boolean`, vaut `false`. Si elle vaut `true`, le fond prendra la couleur indiquée par la propriété `FillColor`. Dans ce cas, la transparence peut dépendre d'un dégradé de transparence.

```
rem Code13-02.sxd   bibli : Hachures Module3
Option Explicit

Sub FondDesHachures()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim servHach As Object, nomHach As String
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")
if MsgBox("Fond transparent ?", 4) = 6 then ' fond transparent
  maForme.FillBackground = false
else ' la couleur du fond est FillColor
  maForme.FillBackground = true
end if
End Sub
```

Fond à motif bitmap

Nous retrouvons ici des notions similaires à celles développées pour les dégradés de transparence. Particularités :

- Le service de motifs *bitmap* a pour nom `com.sun.star.drawing.BitmapTable`.
- La propriété de nom de motif *bitmap* s'intitule `FillBitmapName`.
- La propriété contenant le nom de *bitmap* d'une forme a pour nom `FillBitmap`, de type `String`.
- L'utilisateur n'a pas accès au nom du motif bitmap.

Comme pour les dégradés de transparence, nous utiliserons une macro pour définir des noms de motifs personnels, qui seront mémorisés dans un document modèle.

```
rem Code13-02.sxd   bibli : Motifbmp Module1
Option Explicit

Sub NommerMotifBitmap()
Dim monDocument As Object, maPage As Object
Dim maForme As Object, nouvNom As String
Dim servMotifs As Object, nomMotif As String, unMotif As String
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")

nomMotif = maForme.FillBitmapName
servMotifs = monDocument.createInstance(_
           "com.sun.star.drawing.BitmapTable")
if servMotifs.hasByName(nomMotif) then
  unMotif = servMotifs.getByName(nomMotif)
  nouvNom = InputBox("Nom du nouveau motif bitmap")
  if nouvNom <> "" then servMotifs.insertByName(nouvNom,unMotif)
else
  MsgBox("Nom de motif inconnu : " & nomMotif, 16)
end if
End Sub
```

Ici aussi, la méthode `removeByName` permet de supprimer un motif en connaissant son nom.

Un motif *bitmap* est appliqué à une forme en copiant son nom dans la propriété `FillBitmapName` et en affectant à la propriété `FillStyle` la constante nommée `BITMAP`.

```
rem Code13-02.sxd   bibli : Motifbmp Module3
Option Explicit

Sub AppliquerMotifBitmap()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim servMotifs As Object, nomMotif As String, unMotif As String
```

```
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F2")

nomMotif = "maFlamme"
servMotifs = monDocument.createInstance(_
           "com.sun.star.drawing.BitmapTable")
if servMotifs.hasByName(nomMotif) then
  maForme.FillBitmapName = nomMotif
  maForme.FillStyle = com.sun.star.drawing.FillStyle.BITMAP
else
  MsgBox("Nom de motif inconnu : " & nomMotif, 16)
end if
End Sub
```

On peut ajouter un dégradé de transparence.

L'ombre d'une forme

L'ombre portée par une forme est définie par plusieurs propriétés, listées au tableau 13-10.

Tableau 13–10 Propriétés d'ombre de forme

Propriété	Type	Signification
Shadow	Boolean	True pour activer l'ombre.
ShadowColor	Long	Couleur de l'ombre.
ShadowTransparence	Integer	Pourcentage de transparence, entre 0 (opaque) et 100 (transparent).
ShadowXDistance	Long	Décalage horizontal de l'ombre, mesuré par rapport au bord gauche de la forme ; en 1/100 de mm, positif vers la droite.
ShadowYDistance	Long	Décalage vertical de l'ombre, mesuré par rapport au bord du haut de la forme ; en 1/100 de mm, positif vers le bas.

Les décalages de l'ombre peuvent avoir une valeur négative.

```
rem Code13-02.sxd   bibli : Proprietes Module3
Option Explicit

Sub OmbreForme()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin3")
maForme = FindObjectByName(maPage, "F5")
```

```
With maForme
  .ShadowColor = RGB(100,100,100)
  .ShadowTransparence = 30
  .ShadowXDistance = -1000 ' ombre portée à gauche
  .ShadowYDistance = 300   ' et vers le bas
  .Shadow = true
End With
End Sub
```

Angle de rotation de la forme

Cette propriété effectue une rotation de la forme.

```
maForme.RotateAngle = 2500 ' 25 degrés
```

L'angle est exprimé en 1/100 de degré par rapport à l'horizontale ; une valeur positive correspond au sens *inverse* des aiguilles d'une montre.

Cisaillement de la forme

Cet effet transforme un rectangle en parallélogramme. Pour un autre type de forme, l'effet est équivalent.

```
maForme.ShearAngle = 3000 ' 30 degrés de cisaillement
```

L'angle est exprimé en 1/100 de degré par rapport au côté vertical d'un rectangle ; une valeur positive correspond au sens des aiguilles d'une montre.

Écrire un texte dans une forme

Le texte d'une forme peut être géré dans sa totalité, mais il est aussi possible de modifier le formatage de certains caractères du texte.

Gestion globale du texte

Pour un usage simple, on peut se contenter de l'objet `Text` exposé par la forme. L'ensemble du texte est disponible en lecture et en écriture dans la propriété `String`, de type `String`, de l'objet texte. Cet exemple écrit un texte dans une forme nommée `F1`.

```
rem Code13-03.sxd   bibli : Texte Module1
Option Explicit
```

```
Sub EcrireTexteGlobal()
Dim monDocument As Object, maPage As Object, maForme As Object
Dim monTexte As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")
maForme = FindObjectByName(maPage, "F1")

monTexte = maForme.Text
monTexte.String = "Bonjour !"
End Sub
```

On peut appliquer au texte dans son ensemble la plupart des propriétés de formatage des caractères détaillées au chapitre 11 sur Writer, section « Formatage local ». Le tableau 13-11 liste les propriétés qui ne sont pas reconnues.

Tableau 13–11 Propriétés de caractère non reconnues

Propriété	Signification
CharBackColor	Couleur du fond.
CharBackTransparent	Transparence du fond.
CharCaseMap	Changement de casse.
CharStyleName	Nom du style de caractère.

Il suffit de reprendre dans les exemples pour Writer la ligne de formatage en remplaçant `monCurseur` par `monTexte`. À titre d'exemple, ce codage modifie la taille des caractères du texte de la forme.

```
rem Code13-03.sxd   bibli : Texte Module2
Option Explicit

Sub TaillePoliceTexte()
Dim monDocument As Object, maPage As Object, maForme As Object
Dim monTexte As Object, taille As Long
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")
maForme = FindObjectByName(maPage, "F1")
monTexte = maForme.Text

Taille = InputBox("Taille de la police (en points)")
if Taille > 0 then
  monTexte.CharHeight = Taille
end if
End Sub
```

Notez que pour définir le formatage de caractère sur une forme qui n'a pas encore reçu de texte, vous devez utiliser le curseur d'écriture.

Position du texte dans la forme

Elle dépend des propriétés indiquées au tableau 13-12, qui correspondent aux paramètres du panneau contextuel Texte, onglet Texte reproduit sur la figure 13-1.

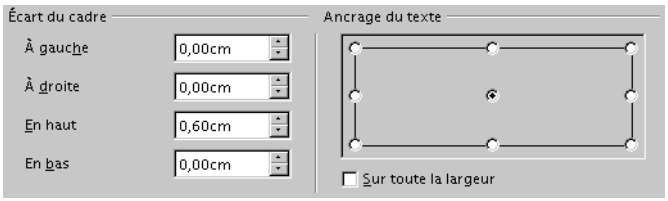


Figure 13–1
Panneau contextuel Texte sur une forme

Tableau 13–12 Propriétés de position du texte

Propriété	Type	Signification
`TextHorizontalAdjust`	`Long`	Point d'origine de la position horizontale.
`TextLeftDistance`	`Long`	Écart du cadre à gauche du texte.
`TextRightDistance`	`Long`	Écart du cadre à droite du texte.
`TextVerticalAdjust`	`Long`	Point d'origine de la position verticale.
`TextUpperDistance`	`Long`	Écart du cadre en haut du texte.
`TextLowerDistance`	`Long`	Écart du cadre en bas du texte.
`TextFitToSize`	`Long`	Adaptation de la police au cadre du texte ; constante nommée.
`TextContourFrame`	`Boolean`	`True` pour adapter le texte au contour.

L'ancrage du texte est la combinaison des propriétés `TextHorizontalAdjust` et `TextVerticalAdjust`. Elles reçoivent respectivement des constantes nommées de la forme :

```
com.sun.star.drawing.TextHorizontalAdjust.CENTER
com.sun.star.TextVerticalAdjust.CENTER
```

Les valeurs possibles sont listées au tableau 13-13 (quatre valeurs pour chaque propriété).

Tableau 13–13 Constantes de position de texte

Constante	Position du texte
`LEFT`	Horizontalement : à gauche.
`CENTER`	Horizontalement : au centre.
`RIGHT`	Horizontalement : à droite.
`BLOCK`	Horizontalement : étalé
`TOP`	Verticalement : en haut
`CENTER`	Verticalement : au centre
`BOTTOM`	Verticalement : en bas
`BLOCK`	Verticalement : étalé

La propriété `TextFitToSize` reçoit une constante nommée de la forme :

```
com.sun.star.drawing.TextFitToSizeType.PROPORTIONAL
```

Les différentes valeurs sont listées au tableau 13-14. Cependant, seules les valeurs `NONE` et `PROPORTIONAL` semblent avoir une utilité, contrairement à ce que dit la documentation API.

Tableau 13–14 Constantes d'adaptation de la police au cadre

Constante	Adaptation
NONE	Pas d'adaptation.
PROPORTIONAL	La largeur de la police est modifiée pour que la plus longue ligne du texte corresponde à la largeur du rectangle englobant la forme. La hauteur de la police dépend de la hauteur du rectangle englobant.
ALLLINES	Même effet que PROPORTIONAL ?
RESIZEATTR	Pas d'effet visible d'adaptation.

Texte animé

Ces propriétés, qui s'appliquent à l'ensemble du texte de la forme, sont disponibles pour Draw mais en pratique utilisées dans le contexte d'une présentation Impress visualisée avec un ordinateur. Elles sont listées dans le tableau 13-15.

Tableau 13–15 Propriétés d'animation du texte

Propriété	Type	Signification
TextAnimationKind	Long	Genre d'animation ; constante nommée.
TextAnimationDirection	Long	Sens du défilement du texte ; constante nommée.
TextAnimationAmount	Integer	Nombre de pixels de décalage à chaque pas du défilement.
TextAnimationDelay	Integer	Nombre de millisecondes entre chaque pas du défilement, ou pour chaque étape du clignotement.
TextAnimationCount	Integer	Nombre de répétitions de l'animation ; zéro s'il n'y a pas de limitation.
TextAnimationStartInside	Boolean	True pour que le texte soit visible au début de l'animation.
TextAnimationStopInside	Boolean	True pour que le texte soit visible à la fin de l'animation.

Le tableau 13-16 liste les valeurs possibles de la propriété `TextAnimationKind`. Les constantes nommées sont de la forme :

```
com.sun.star.drawing.TextAnimationKind.SCROLL
```

Tableau 13–16 Constantes de genre d'animation du texte

Constante	Résultat
NONE	Pas d'animation.
BLINK	Texte clignotant.
SCROLL	Texte défilant.
ALTERNATE	Texte défilant alternativement dans un sens puis dans l'autre.
SLIDE	Le texte apparaît en défilant et s'arrête une fois en place. La propriété `TextAnimationCount` doit être à zéro.

Le tableau 13-17 indique les différentes directions de défilement. Ces constantes nommées sont de la forme :

```
com.sun.star.drawing.TextAnimationDirection.UP
```

Tableau 13–17 Constantes de direction de défilement

Constante	Défilement du texte
LEFT	Vers la gauche.
RIGHT	Vers la droite.
UP	Vers le haut.
DOWN	Vers le bas.

EXERCICE

Nous laisserons le lecteur s'amuser à tester toutes ces possibilités avec une macro.

Utilisation d'un curseur d'écriture

Pour formater certains caractères du texte différemment des autres, nous aurons besoin d'un curseur d'écriture qui permet de pointer sur une zone dans le texte. Nous ne développerons pas ici cette méthode, qui est expliquée au chapitre 11 sur Writer, dans les sections « Le curseur d'écriture », « Insérer du texte » et « Formatage local des caractères ». Le curseur d'écriture obtenu est un simple curseur de caractère, il ne reconnaît pas les mots ni les phrases. Le formatage de caractères a les mêmes limitations que celles listées plus haut, voir tableau 13-11.

Dans cet exemple, nous allons modifier le texte « Bonjour ! » que nous avons écrit précédemment dans la forme `F1`.

```
rem Code13-03.sxd   bibli : Texte Module3
Option Explicit

Sub ManipulerTexte()
Dim monDocument As Object, maPage As Object, maForme As Object
Dim monTexte As Object, monCurseur As Object, pargr As String
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")
maForme = FindObjectByName(maPage, "F1")

pargr = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
monTexte = maForme.Text
monCurseur = monTexte.createTextCursor
With monCurseur
  .gotoStart(false) ' ne pas oublier de positionner le curseur
  .goRight(1, true) 'sélectionner le 1er caractère
  .CharColor = RGB(255,0,0) ' peindre en rouge
  .CharWeight = com.sun.star.awt.FontWeight.BOLD
  .gotoEnd(false) ' aller en fin de texte
  .goLeft(1, false) ' se placer avant le point d'exclamation
  monTexte.insertString(monCurseur, "tout le monde ", false)
  .gotoEnd(false) ' aller en fin de texte
  ' insérer une fin de paragraphe
  monTexte.insertControlCharacter(monCurseur, pargr, false)
  .CharHeight = 16 ' changer la taille à partir d'ici
  .CharFontName = "Arial" ' changer la police
  .CharUnderlineHasColor = false
  .CharUnderline = com.sun.star.awt.FontUnderline.DOUBLE
  monTexte.insertString(monCurseur, "OpenOffice.org", false)
End With
End Sub
```

Pour éviter des problèmes d'initialisation, positionnez le curseur en début ou fin de texte, juste après sa création.

Vous avez remarqué l'insertion d'un paragraphe. Effectivement, le curseur propose un service restreint de paragraphe. Les propriétés offertes sont :

- `ParaAdjust`,
- `ParaTopMargin`,
- `ParaBottomMargin`,
- `ParaLeftMargin`,
- `ParaRightMargin`,
- `ParaFirstLineIndent`,
- `ParaLastLineAdjust`,
- `ParaLineSpacing`,
- `ParaTabStops`.

Les différentes formes

Nous avons vu plus haut comment ajouter une forme sur une page. Le paramètre de la méthode `createInstance` précise quel genre de forme doit être créé. Le tableau 13-18 liste les valeurs possibles, qui sont des chaînes de caractères représentant des noms de service ; leur nom complet, dont la casse est à respecter, est de la forme :

```
com.sun.star.drawing.RectangleShape
```

Tableau 13–18 Noms des formes

Nom	Forme
`RectangleShape`	Rectangle ou carré.
`EllipseShape`	Ellipse ou cercle.
`LineShape`	Ligne droite.
`PolyLineShape`	Ligne brisée.
`PolyPolygonShape`	Polygone.
`TextShape`	Texte.
`CaptionShape`	Étiquette.
`ConnectorShape`	Connecteur.
`MeasureShape`	Ligne de cote.
`OpenBezierShape`	Courbe de Bézier ouverte.
`ClosedBezierShape`	Courbe de Bézier fermée.
`PolyPolygonBezierShape`	Polygone de Bézier.

Nous ne traiterons pas des objets 3D dans ce document, faute de documentation sur ce sujet dans le SDK.

Le rectangle, le carré

Le rectangle a déjà été utilisé quand nous avons créé une forme. Si la hauteur est égale à la largeur, évidemment vous obtenez un carré.

La propriété `CornerRadius`, de type `Long`, indique le rayon, mesuré en 1/100 de millimètres, du quart de cercle effectué à chaque coin. Une valeur nulle produit un rectangle ordinaire, une autre valeur produit un rectangle aux coins arrondis.

L'ellipse, le cercle

Si la hauteur de l'ellipse est égale à sa largeur, vous obtenez un cercle.

La propriété `CircleKind`, de type `Long`, permet d'obtenir différentes variantes d'ellipse, listées au tableau 13-19. Cette propriété reçoit une constante nommée de la forme :

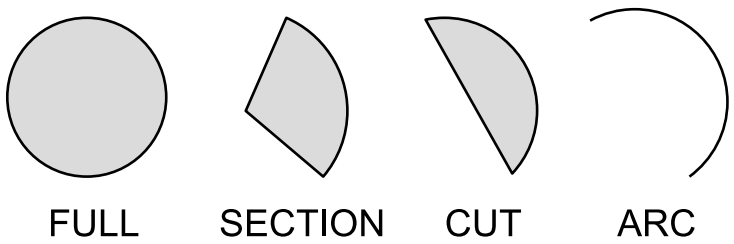
```
com.sun.star.drawing.CircleKind.SECTION
```

Tableau 13–19 Constantes de variante d'ellipse

Constante	Signification
FULL	Disque.
SECTION	Tranche de camembert.
CUT	Segment de cercle délimité par l'arc de cercle et sa corde.
ARC	Arc de cercle

La figure 13-2 représente les différentes formes possibles.

Figure 13–2
Les variantes d'ellipse



Le positionnement de l'arc de cercle dans les variantes dépend de deux autres propriétés, de type `Long` :

- `CircleStartAngle` est l'angle initial de l'arc.
- `CircleEndAngle` est l'angle terminal de l'arc.

Les angles sont mesurés en 1/100 de degré, valeurs positives dans le sens inverse des aiguilles d'une montre. La position zéro de l'angle est l'horizontale, avec le centre de rotation situé à gauche de l'arc.

Le texte

La forme de type texte peut être dessinée avec un contour apparent, ce qui donne un cadre. Dans ce cas, il est possible de le forcer à s'adapter au texte qu'il contient, grâce aux propriétés listées au tableau 13-20.

L'équivalent sur l'interface utilisateur se trouve dans le panneau Position et taille, onglet Position et taille, cases à cocher Adapter la largeur au texte et Adapter la hauteur au texte.

Tableau 13–20 Propriétés d'adaptation automatique du cadre

Propriété	Type	Signification
TextAutoGrowHeight	Boolean	True pour que la hauteur du cadre s'adapte au texte.
TextMaximumFrameHeight	Long	Hauteur maximale autorisée, en 1/100 de mm.
TextMinimumFrameHeight	Long	Hauteur minimale autorisée, en 1/100 de mm.
TextAutoGrowWidth	Boolean	True pour que la largeur du cadre s'adapte au texte.
TextMaximumFrameWidth	Long	Largeur maximale autorisée, en 1/100 de mm.
TextMinimumFrameWidth	Long	Largeur minimale autorisée, en 1/100 de mm.

Dans le document du zip téléchargeable, nous avons mis dans la page Dessin2 une forme texte F5 avec un cadre apparent. Cet exemple montre l'effet sur la largeur, selon les textes à afficher.

```
rem Code13-03.sxd   bibli : Texte Module4
Option Explicit

Sub EnveloppeElastique()
Dim monDocument As Object, maPage As Object, maForme As Object
Dim nouvTxt As String, souhaitAuto As Long
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin2")
maForme = FindObjectByName(maPage, "F5")
Do
  souhaitAuto = MsgBox("Largeur automatique ?", 3)
  if souhaitAuto = 2 then exit do ' réponse = Annuler
  maForme.TextAutoGrowWidth = (souhaitAuto = 6)
  maForme.String = InputBox("Nouveau texte", "", maForme.String)
Loop
End Sub
```

La ligne simple

Rien n'est plus simple que de définir une ligne simple avec une forme de type LineShape : la position et les dimensions de la forme suffisent. Cependant, il existe une deuxième méthode pour la définir, en considérant qu'elle n'est rien d'autre qu'une ligne brisée à un seul segment.

Les propriétés de contour s'appliquent à la ligne (qu'elle soit simple, brisée, ou autre). Les extrémités d'une ligne (flèche, carré, rond) sont des poly-polygones de Bézier rattachés au début et à la fin de la ligne. Ces objets sont accessibles par les propriétés LineStart et LineEnd respectivement. Il n'est pas possible d'obtenir des extrémités prédéfinies, au contraire de l'interface utilisateur.

Notez enfin qu'il est parfaitement possible d'affecter un texte à une ligne, et d'ajuster la position du texte pour le placer au-dessus ou au-dessous de la ligne.

La ligne brisée

Une ligne brisée (`PolyLineShape`) est décrite à partir des coordonnées des points extrémités des segments de droite successifs. Chaque point est défini avec une structure dont les composants X et Y sont respectivement les distances horizontale et verticale par rapport au coin haut-gauche de la page. Ces distances sont mesurées en 1/100 de millimètres. À la série de points correspondra un tableau de structures.

Nous devons d'abord ajouter la ligne brisée à la page, sans indication de position ni dimension. Ensuite seulement, nous affecterons à la forme les coordonnées de points.

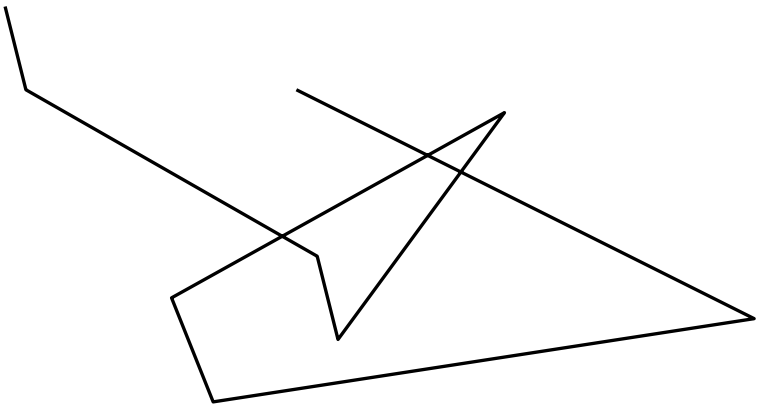
```
rem Code13-04.sxd   bibli : Polygones Module2
Option Explicit

Sub LigneBrisee()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim lesPoints(8) As New com.sun.star.awt.Point
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Polygones")
maForme = monDocument.createInstance(_
          "com.sun.star.drawing.PolyLineShape")
lesPoints(0).X =  4000
lesPoints(0).Y =  2000
lesPoints(1).X =  4500
lesPoints(1).Y =  4000
lesPoints(2).X = 11500
lesPoints(2).Y =  8000
lesPoints(3).X = 12000
lesPoints(3).Y = 10000
lesPoints(4).X = 16000
lesPoints(4).Y =  4550
lesPoints(5).X =  8000
lesPoints(5).Y =  9000
lesPoints(6).X =  9000
lesPoints(6).Y = 11500
lesPoints(7).X = 22000
lesPoints(7).Y =  9500
lesPoints(8).X = 11000
lesPoints(8).Y =  4000
maPage.add(maForme)
maForme.LineWidth = 80
maForme.PolyPolygon = Array(lesPoints())
End Sub
```

La manière d'affecter les points est particulière : l'utilisation de la fonction Basic `Array()` constitue un tableau dont les éléments sont les arguments. Ici, il y a un seul élément, `lesPoints()` qui est lui-même notre tableau de points. La raison d'être de cette construction est que la ligne brisée utilise des mécanismes qui servent aussi à dessiner des poly-

gones multiples. La ligne brisée est considérée comme un polygone unique et ouvert. La forme résultante est entièrement dépendante des points successifs, comme on le voit sur la figure 13-3.

Figure 13–3
Une ligne brisée



Nous avons dit que la ligne simple était un cas particulier de la ligne brisée. En effet, pour la construire il suffirait de reprendre le code ci-dessus avec le tableau `lesPoints()` réduit à deux éléments et de créer la forme avec le service `LineShape` au lieu de `PolyLineShape`.

Le polygone

Reprenons l'exemple de la ligne brisée. Il suffit d'utiliser le service `PolyPolygonShape` au lieu du service `PolyLineShape` pour obtenir un polygone fermé. Le point initial et le point final sont automatiquement reliés par un segment de droite.

```
rem Code13-04.sxd   bibli : Polygones Module3
Sub unPolygone()
' - - - - - - - partie identique - - - - - -
maForme = monDocument.createInstance(_
          "com.sun.star.drawing.PolyPolygonShape")
' - - - - - - - partie identique - - - - - -
```

Le dessin obtenu, reproduit figure 13-4, est étonnant en ce qui concerne le remplissage.

La figure contient des trous lorsque les lignes se croisent. Il s'agit bien de trous, car une autre forme en arrière-plan apparaîtrait dans ces zones. Quel est le principe de « création » de ces trous ? OpenOffice.org part d'un point à l'extérieur du dessin, et analyse par exemple de gauche à droite. À partir de la première ligne du dessin rencontrée, OpenOffice.org décide qu'il se trouve sur la surface du dessin ; s'il rencontre une autre ligne du même dessin, il décide qu'il vient de passer à l'extérieur. Le processus se répète à chaque ligne traversée. Les trous sont les zones considérées comme extérieures au polygone fermé.

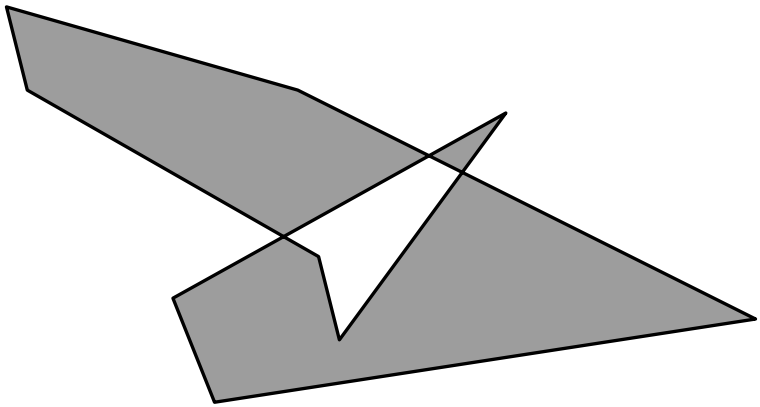


Figure 13–4
Polygone fermé

Nous pouvons donc créer des formes sans trous ou avec trous, mais pas n'importe lesquels : nous ne pouvons pas créer un trou sans aucun point commun avec le contour englobant. Pour plus de liberté nous avons besoin du poly-polygone.

Le poly-polygone

Le poly-polygone consiste à utiliser plusieurs polygones dans la suite de points définissant la forme. Certains polygones peuvent être inclus à l'intérieur d'autres. Cependant, comme ils sont considérés comme appartenant à la même forme, l'algorithme de décision surface/trou s'appliquera à chaque ligne rencontrée.

Nous allons dessiner un rectangle avec deux trous en triangles opposés par un sommet. Cette fois-ci la propriété `PolyPolygon` recevra un tableau de deux éléments, le premier étant le rectangle, le deuxième étant un polygone croisé.

```
rem Code13-04.sxd   bibli : Polygones Module4
Option Explicit

Sub GruyereAbstrait()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim enveloppe(3) As New com.sun.star.awt.Point
Dim triangles(3) As New com.sun.star.awt.Point
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Polygones")
maForme = monDocument.createInstance(_
          "com.sun.star.drawing.PolyPolygonShape")
```

```
enveloppe(0).X =  4500
enveloppe(0).Y =  3500
enveloppe(1).X =  4500
enveloppe(1).Y =  9500
enveloppe(2).X = 19000
enveloppe(2).Y =  9500
enveloppe(3).X = 19000
enveloppe(3).Y =  3500
triangles(0).X =  6000
triangles(0).Y =  5000
triangles(1).X = 17000
triangles(1).Y =  9000
triangles(2).X = 17000
triangles(2).Y =  5000
triangles(3).X =  6000
triangles(3).Y =  9000
maPage.add(maForme)
maForme.LineWidth = 80
maForme.PolyPolygon = Array(enveloppe(), triangles())
End Sub
```

La forme résultante est reproduite à la figure 13-5. Dans le document du zip téléchargeable, la forme se superpose à un dessin existant, afin de mettre en valeur les trous.

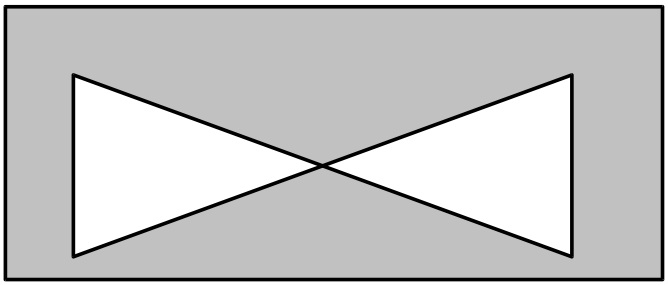


Figure 13–5
Poly-Polygone troué

Le connecteur

Les propriétés de ligne simple sont aussi disponibles pour le connecteur. Les symboles éventuels (flèche, carré, rond) à une extrémité du connecteur sont des poly-polygones de Bézier rattachés au début et à la fin de la ligne. Ces objets sont accessibles par les propriétés `LineStart` et `LineEnd`.

Les principales propriétés spécifiques du connecteur sont listées au tableau 13-21. Les huit premières correspondent au panneau `Connecteur` du menu contextuel, dans l'interface utilisateur.

Tableau 13–21 Propriétés d'un connecteur

Propriété	Type	Signification
EdgeKind	Long	Type de connecteur ; constante nommée.
EdgeLine1Delta	Long	Décalage de ligne 1, en 1/100 de mm.
EdgeLine2Delta	Long	Décalage de ligne 2, en 1/100 de mm.
EdgeLine3Delta	Long	Décalage de ligne 3, en 1/100 de mm.
EdgeNode1HorzDist	Long	Écart entre les lignes, début horizontal, en 1/100 de mm.
EdgeNode1VertDist	Long	Écart entre les lignes, début vertical, en 1/100 de mm.
EdgeNode2HorzDist	Long	Écart entre les lignes, fin horizontale, en 1/100 de mm.
EdgeNode2VertDist	Long	Écart entre les lignes, fin verticale, en 1/100 de mm.
StartShape	Object	La forme à laquelle le début du connecteur est connecté ; null si le début du connecteur n'est pas connecté.
StartGluePointIndex	Long	Rang du point de colle auquel le début du connecteur est connecté.
StartPosition	Object	Coordonnées du début du connecteur ; structure de point ; en lecture seule si le connecteur est connecté.
EndShape	Object	Comme StartShape, pour la fin du connecteur.
EndGluePointIndex	Long	Comme StartGluePointIndex pour la fin du connecteur.
EndPosition	Object	Comme StartPosition, pour la fin du connecteur.

Les constantes nommées de la propriété EdgeKind, listées au tableau 13-22, sont de la forme :

```
com.sun.star.drawing.ConnectorType.STANDARD
```

BOGUE Constante ConnectorType

La constante LINE est actuellement indisponible dans Basic (OpenOffice.org version 1.1.1). Utilisez directement la valeur numérique 2.

Tableau 13–22 Constantes de type de connecteur

Constante	Type de connecteur
STANDARD	Standard.
CURVE	Incurvé.
LINE (valeur = 2)	Direct.
LINES	Linéaire.

Nous donnons un exemple d'utilisation de connecteur dans la section « Collages (les points de colle) ».

L'étiquette

Le tableau 13-23 liste les propriétés spécifiques essentielles d'une étiquette. Certaines propriétés se retrouvent dans le panneau contextuel Position et taille, onglet Légende. Notez que la propriété `CornerRadius`, déjà vue pour le rectangle, permet d'arrondir les angles du cadre de l'étiquette.

Les symboles éventuels (flèche, carré, rond) à une extrémité du trait indicateur sont des poly-polygones de Bézier rattachés au début et à la fin de la ligne. Ces objets sont accessibles par les propriétés `LineStart` et `LineEnd`.

Tableau 13–23 Propriétés d'une étiquette

Propriété	Type	Signification
`CaptionPoint`	`Object`	Position du point indiqué par l'étiquette ; structure de `Point`, avec des coordonnées relatives à la position du coin haut-gauche de l'étiquette.
`CaptionType`	`Integer`	Type du trait indicateur, constante nommée.
`CaptionGap`	`Long`	Écart entre le trait indicateur et le cadre de l'étiquette.
`CaptionIsEscapeRelative`	`Boolean`	`True` si la position de départ du trait indicateur est exprimée par `CaptionEscapeRelative` ; `False` si elle est exprimée par `CaptionEscapeAbsolute`.
`CaptionEscapeRelative`	`Long`	Position relative au coin haut-gauche de l'étiquette, exprimée en 1/100 de pourcentage de la largeur ou la hauteur de celle-ci. Par exemple, 5 000 correspond à 50,00 % de la largeur ou de la hauteur.
`CaptionEscapeAbsolute`	`Long`	Position relative au coin haut-gauche de l'étiquette, exprimée en 1/100 de mm.
`CaptionEscapeDirection`	`Long`	Direction du trait indicateur, constante nommée.
`CaptionIsFitLineLength`	`Boolean`	`True` pour une longueur optimale du trait indicateur ; `False` si la longueur est imposée par `CaptionIsFitLineLength`.
`CaptionIsFitLineLength`	`Long`	Longueur du premier segment du trait indicateur, en 1/100 de mm.

Le type de trait indicateur, décrit au tableau 13-24, est une constante nommée de la forme :

```
com.sun.star.drawing.CaptionType.connector
```

Tableau 13–24 Type de trait indicateur

Constante	Type de trait
`straight`	Ligne droite.
`angled`	Ligne en angle.
`connector`	Ligne en angle fléchie.

La direction du trait indicateur est réalisée différemment de ce qui est visible dans le panneau d'interface utilisateur. Les valeurs possibles sont listées au tableau 13-25, les constantes nommées sont de la forme :

```
com.sun.star.drawing.CaptionEscapeDirection.horizontal
```

Tableau 13–25 Direction du trait indicateur

Constante	Direction
`horizontal`	Horizontale.
`vertical`	Verticale.
`auto`	Optimale.

La ligne de cote

Le tableau 13-26 liste les propriétés spécifiques essentielles d'une ligne de cote. Elles ont pour la plupart un équivalent dans le panneau contextuel Cotation de l'interface utilisateur. Toutes les longueurs et distances sont exprimées en 1/100 de millimètres sur la page de dessin. Faites des essais sur diverses lignes de cotes pour visualiser les effets (voir le zip téléchargeable, `Code13-03.sxd`, bibliothèque `Cotations`).

Tableau 13–26 Propriétés d'une ligne de cote

Propriété	Type	Signification
`MeasureHelpLine1Length`	`Long`	Longueur du repère gauche.
`MeasureHelpLine2Length`	`Long`	Longueur du repère droit.
`MeasureHelpLineDistance`	`Long`	Écart des repères.
`MeasureHelpLineOverhang`	`Long`	Dépassement des repères.
`MeasureLineDistance`	`Long`	Écart des lignes.
`MeasureTextHorizontalPosition`	`Long`	Position horizontale du texte, constante nommée.
`MeasureTextVerticalPosition`	`Long`	Position horizontale du texte, constante nommée.
`MeasureBelowReferenceEdge`	`Boolean`	`True` pour une ligne de cote sous l'objet.
`MeasureUnit`	`Long`	Unité de mesure, voir tableau 13-29.
`MeasureShowUnit`	`Boolean`	`True` pour afficher l'unité de mesure.

Tableau 13–26 Propriétés d'une ligne de cote (suite)

Propriété	Type	Signification
MeasureTextRotate90	Boolean	True pour mettre le texte perpendiculaire à la ligne de cote. La valeur False correspond à la case Parallèle au repère de l'interface utilisateur.
MeasureTextUpsideDown	Boolean	True pour tourner le texte de 180°.
MeasureDecimalPlaces	Integer	Nombre de décimales de la mesure.

Les positions du texte horizontale et verticale sont listées dans les tableaux 13-27 et 13-28. Les constantes nommées respectives sont de la forme :

```
com.sun.star.drawing.MeasureTextHorzPos.LEFTOUTSIDE
com.sun.star.drawing.MeasureTextVertPos.WEST
```

Tableau 13–27 Position horizontale du texte de cotation

Constante	Position
AUTO	Automatiquement horizontal.
LEFTOUTSIDE	À gauche.
INSIDE	Au milieu.
RIGHTOUTSIDE	À droite.

Tableau 13–28 Position verticale du texte de cotation

Constante	Position
AUTO	Automatiquement vertical.
EAST	Au-dessus de la ligne de cote.
BREAKEDLINE	Au milieu et interrompant la ligne de cote.
WEST	Au-dessous de la ligne de cote.
CENTERED	Au milieu et interrompant la ligne de cote.

Tableau 13–29 Unités de mesure d'une cote

Valeur	Unité
0	Celle du document.
1	mm
2	cm
3	m
4	km
5	Twip
6	Pouce

Tableau 13–29 Unités de mesure d'une cote (suite)

Valeur	Unité
7	Pied
8	Mile
9	Pica
10	Point
11	Non utilisé
12	Pourcentage
13	1/100 de mm

Les formes de Bézier

Nous abordons ici des formes complexes, aussi bien à l'interface utilisateur qu'au niveau de la programmation.

> **À RETENIR Bézier**
>
> Les courbes de Bézier sont des courbes polynomiales décrites pour la première fois en 1972 par l'ingénieur français Pierre Bézier (1910-1999), qui les utilisa pour concevoir par ordinateur des voitures automobiles. Les plus importantes courbes de Bézier sont les cubiques, qui sont utilisées en informatique pour le graphisme et dans de multiples systèmes de traitement d'image tels que PostScript, Metafont et GIMP pour dessiner des courbes «lisses» joignant des points ou des polygones de Bézier. Les fontes TrueType utilisent des courbes de Bézier quadratiques plus simples.
>
> ▸ Source : http://fr.wikipedia.org/

La propriété `PolyPolygonBezier` est une structure de deux éléments :

- `Coordinates` est un tableau de tableaux de points, comme nous en avons vu avec les lignes brisées, les polygones et poly-polygones.
- `Flags` est un tableau de tableaux d'entiers `Long`, tel que chaque élément correspond au point de même index dans `Coordinates`.

Chaque élément de `Flags` contient une constante nommée (tableau 13-30) de la forme :

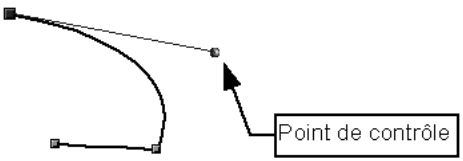
```
com.sun.star.drawing.PolygonFlags.NORMAL
```

Tableau 13–30 Constantes de point de Bézier

Constante	Signification
NORMAL	Point d'inflexion normal.
SMOOTH	Point d'inflexion à jonction lisse.
SYMMETRIC	Point d'inflexion à jonction symétrique.
CONTROL	Point de contrôle.

Lorsque vous éditez un point d'une courbe de Bézier, la barre d'outils affiche son type : inflexion normale, jonction lisse et jonction symétrique. Un point de contrôle est le point d'extrémité de la tangente à la courbe (voir la figure 13-6) ; quand vous déplacez ce point, un effet de levier modifie la forme de la courbe.

Figure 13–6
Courbe et point de contrôle



Si nous utilisions seulement des points d'inflexion, nous obtiendrions une ligne brisée. Pour dessiner une courbe entre deux points A et B, nous avons besoin d'ajouter deux points de contrôle cA et cB ; la position du point cA règle la tangente aboutissant au point A et la position du point cB règle la tangente aboutissant au point B. En résumé, chaque courbe élémentaire est définie par deux points d'inflexion situés aux extrémités de la courbe et deux points de contrôle situés en dehors de celle-ci. Si le point B est une jonction lisse, la courbe gardera une transition douce de part et d'autre du point.

Pour réaliser cela par programmation, il est nécessaire de remplir les propriétés `Coordinates` et `Flags` avec des structures complètes, en décrivant les points d'inflexion ou de contrôle rencontrés successivement, par exemple : A, cA, cB, B, C. N'oubliez pas que les points de contrôle sont utilisés par paires.

```
rem Code13-04.sxd   bibli : Polygones Module5
Option Explicit

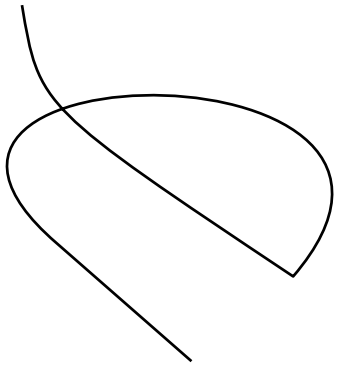
Sub LigneBezier()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
Dim lesPoints(7) As New com.sun.star.awt.Point
Dim lesInflex(7) As Long
Dim BezCoo As New com.sun.star.drawing.PolyPolygonBezierCoords
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Polygones")
maForme = monDocument.createInstance(_
          "com.sun.star.drawing.OpenBezierShape")
lesPoints(0).X =  3500
lesPoints(0).Y =  2000
lesInflex(0) = com.sun.star.drawing.PolygonFlags.NORMAL
lesPoints(1).X =  4000
lesPoints(1).Y =  5000
lesInflex(1) = com.sun.star.drawing.PolygonFlags.CONTROL
lesPoints(2).X =  4000
lesPoints(2).Y =  5000
lesInflex(2) = com.sun.star.drawing.PolygonFlags.CONTROL
```

```
lesPoints(3).X = 11500
lesPoints(3).Y = 10000
lesInflex(3) = com.sun.star.drawing.PolygonFlags.NORMAL
lesPoints(4).X = 17500
lesPoints(4).Y =  3000
lesInflex(4) = com.sun.star.drawing.PolygonFlags.CONTROL
lesPoints(5).X = -2300
lesPoints(5).Y =  3100
lesInflex(5) = com.sun.star.drawing.PolygonFlags.CONTROL
lesPoints(6).X =  4500
lesPoints(6).Y =  9000
lesInflex(6) = com.sun.star.drawing.PolygonFlags.SMOOTH
lesPoints(7).X =  8500
lesPoints(7).Y = 12500
lesInflex(7) = com.sun.star.drawing.PolygonFlags.NORMAL

BezCoo.Coordinates = Array(lesPoints())
BezCoo.Flags = Array(lesInflex())
maPage.add(maForme)
maForme.LineWidth = 80
maForme.PolyPolygonBezier = BezCoo
End Sub
```

La forme résultante est reproduite à la figure 13-7. En sélectionnant chaque point de la forme, vous pourrez faire apparaître les points de contrôle et vérifier leur position (mesurée par rapport à la page entière).

Figure 13–7
Courbe de Bézier



Une forme de Bézier fermée s'obtient en invoquant le service `ClosedBezierShape` au lieu de `OpenBezierShape`.

EXERCICE

Nous laisserons le lecteur s'exercer à obtenir de jolis poly-polygones de Bézier, en s'inspirant du poly-polygone ordinaire et en utilisant le service `PolyPolygonBezierShape`.

Collages

Les points de colle

Les points de colle d'une forme sont accessibles par sa propriété `GluePoints`. Cet objet est une collection de points de colle, mais un peu particulière. En effet, l'utilisateur peut ajouter (et supprimer) des points de colle à la forme et il faut un moyen pour identifier chaque point, fourni par défaut ou ajouté. L'identificateur de point de colle dans la forme est un simple entier de type `Long`.

La collection `GluePoints` ne peut *pas* être indexée par OOoBasic pour obtenir un point particulier. Il serait possible d'utiliser la fonction `getByIndex` de l'objet collection, mais peu judicieux. Il est préférable d'utiliser à la place la fonction `getByIdentifier`, qui prend en argument l'identificateur du point.

Une forme possède quatre points de colle prédéfinis, identifiés par les numéros 0 à 3. Voici leur position, par rapport au rectangle englobant la forme :

- 0 : en haut et au milieu,
- 1 : à droite et à mi-hauteur,
- 2 : en bas et au milieu,
- 3 : à gauche et à mi-hauteur.

Chaque objet point de colle est une structure détaillée au tableau 13-31. Notez que les explications dans le SDK concernant les propriétés `Position`, `PositionAlignment` et `IsRelative` ne sont pas exactes. Ce qui est décrit ici a été vérifié sur la version 1.1.1 d'OpenOffice.org.

Tableau 13–31 Structure d'un point de colle

Propriété	Type	Signification
`Position`	`Object`	Coordonnées du point de colle par rapport à `PositionAlignment` ; cet objet est une structure de Point.
`IsRelative`	`Boolean`	`True` si la position est donnée en pourcentage.
`PositionAlignment`	`Long`	Origine des coordonnées pour le point de colle ; constante nommée.
`Escape`	`Long`	Direction d'échappement du connecteur collé sur le point ; constante nommée.
`IsUserDefined`	`Boolean`	`True` si le point est ajouté par l'utilisateur.

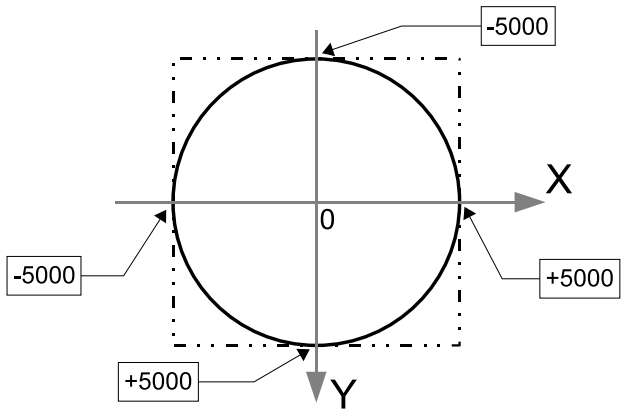
La `Position` est exprimée en coordonnées X et Y par rapport à un point origine. Les valeurs positives vont vers la droite pour X, vers le bas pour Y, et les valeurs négatives vont respectivement vers la gauche et vers le haut (voir figure 13-8).

Le point origine est précisé par la propriété `PositionAlignment`. Les valeurs possibles sont énumérées dans le tableau 13-32 et se réfèrent au rectangle englobant ; les valeurs sont de la forme :

```
com.sun.star.drawing.Alignment.CENTER
```

Si la propriété `IsRelative` vaut `True`, les coordonnées X et Y sont exprimées en centièmes de pourcentage de la largeur totale pour X et de la hauteur totale pour Y. Par exemple, un point situé sur le côté droit d'un rectangle a pour valeur +50,00 %, ce qui correspond à une valeur de 5 000 pour X. La figure 13-8 présente les coordonnées par rapport au centre d'un cercle. Le rectangle englobant est dessiné en tireté.

Figure 13–8
Coordonnées relatives centrées



Si la propriété `IsRelative` vaut `False`, les coordonnées sont exprimées en 1/100 de millimètres, toujours par rapport au point origine.

Tableau 13–32 Constantes de PositionAlignment

Constante	Point d'origine des coordonnées
TOP_LEFT	Le coin haut-gauche.
TOP	Le milieu du côté du haut.
TOP_RIGHT	Le coin haut-droit.
LEFT	Le milieu du côté de gauche.
CENTER	Le centre du rectangle.
RIGHT	Le milieu du côté de droite.
BOTTOM_LEFT	Le coin bas-gauche.
BOTTOM	Le milieu du côté du bas.
BOTTOM_RIGHT	Le coin bas-droit.

Les constantes de la propriété `Escape`, listées au tableau 13-33, sont assez explicites, sauf la première. Forme générale :

```
com.sun.star.drawing.EscapeDirection.RIGHT
```

Tableau 13–33 Constantes de Escape

Constante	Direction
SMART	À l'initiative d'OpenOffice.org.
LEFT	Vers la gauche.
RIGHT	Vers la droite.
UP	Vers le haut.
DOWN	Vers le bas.
HORIZONTAL	Horizontalement (gauche ou droite).
VERTICAL	Verticalement (haut ou bas).

Un point de colle posé par programmation peut se situer ailleurs que sur le contour de la forme, mais à l'intérieur du rectangle englobant.

Le code ci-dessous liste les points de colle d'une forme. Dans le document du zip téléchargeable, nous avons ajouté deux points de colle, dont un est en position relative.

```
rem Code13-05.sxd   bibli : Collages Module1
Option Explicit

Sub DemoPointsDeColle()
  ListerPointsdeColle("carre1")
End Sub

Sub ListerPointsdeColle(nomForme As String)
Dim monDocument As Object, maPage As Object, maForme As Object
Dim lesPtColle As Object, unPtColle As Object
Dim x As Long, idPtColle As Long
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")
maForme = FindObjectByName(maPage, "carre1")
lesPtColle = maForme.GluePoints
for x = 0 to lesPtColle.Count -1
  idPtColle = lesPtColle.Identifiers(x)
  unPtColle = lesPtColle.getByIdentifier(idPtColle)
  print idPtColle, unPtColle.PositionAlignment, _
        unPtColle.IsRelative, unPtColle.IsUserDefined, _
        unPtColle.Position.X, unPtColle.Position.Y
next
End Sub
```

Ajouter un point de colle

Si vous avez bien compris ce qui précède, l'insertion d'un point de colle vous paraîtra limpide. Nous allons insérer un point sur le contour d'un cercle. Connaissant le point X et le rayon du cercle, le théorème de Pythagore nous donne la valeur de Y.

```
rem Code13-05.sxd   bibli : Collages Module2
Option Explicit

Sub AjouterPtColle()
Dim monDocument As Object, maPage As Object, maForme As Object
Dim lesPtColle As Object, idPtColle As Long
Dim unPtColle As New com.sun.star.drawing.GluePoint2
Dim unPoint As New com.sun.star.awt.Point
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")
maForme = FindObjectByName(maPage, "cercle1")
lesPtColle = maForme.GluePoints

unPoint.X = 1600 ' 16% du diamètre = 32% du rayon de la forme
' Le rayon du cercle vaut 50,00% de la hauteur de la forme
' appliquer le théorème de Pythagore
unPoint.Y = sqr(5000*5000 - unPoint.X * unPoint.X)

With unPtColle
  .IsRelative = true ' position en pourcentage
  .PositionAlignment = com.sun.star.drawing.Alignment.CENTER
  .Position = unPoint
  .Escape = com.sun.star.drawing.EscapeDirection.RIGHT
end With
idPtColle = lesPtColle.insert(unPtColle)
print "Identité du nouveau point : " & idPtColle
ListerPointsdeColle("cercle1")
End Sub
```

Nous utilisons une variable objet de type `GluePoint2` (la version `GluePoint` est obsolète). Une fois cet objet initialisé, nous le mettons en argument de la méthode `insert` de l'objet collection de points de colle.

Supprimer un point de colle

Il n'existe pas de fonction `hasByIdentifier` qui permettrait de savoir si un identificateur est présent dans la forme. La méthode `removeByIdentifier` de l'objet collection de points de colle supprime le point fourni en argument. S'il n'existe pas de point ayant cette identité, une exception est déclenchée. Nous allons utiliser cette particularité dans la macro exemple, en interceptant l'exception avec l'instruction `on error Goto`.

```
rem Code13-05.sxd   bibli : Collages Module3
Option Explicit

Sub SupprimerPtColle()
Dim monDocument As Object, maPage As Object, maForme As Object
Dim lesPtColle As Object, idPtColle As Long
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")
maForme = FindObjectByName(maPage, "cercle1")
lesPtColle = maForme.GluePoints

idPtColle = InputBox("Identité du point à supprimer")
on error Goto pointNex ' intercepter les exceptions
lesPtColle.removeByIdentifier(idPtColle)
on error Goto 0
ListerPointsdeColle("cercle1")
exit sub
pointNex: ' removeByIdentifier a déclenché une exception
  on error Goto 0
  MsgBox("Point de colle inexistant ou interdit", 16)
End Sub
```

Vous pourrez vérifier qu'il est interdit de supprimer un des points de colle prédéfinis.

Relier deux formes par un connecteur

Nous allons utiliser les notions de point de colle et les caractéristiques de connecteur dans la macro suivante. Dans le document du zip téléchargeable, nous avons dessiné et nommé deux formes que nous allons relier par un connecteur. Comme le connecteur sera positionné par les points de colle des deux formes, il est inutile de le positionner à sa création.

```
rem Code13-05.sxd   bibli : Collages Module4
Option Explicit

Sub Connecter2Formes()
Dim monDocument As Object, maPage As Object
Dim leConnecteur As Object, uneForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessin1")

leConnecteur = monDocument.createInstance(_
               "com.sun.star.drawing.ConnectorShape")
' inutile de définir la position et les dimensions
' car elles seront imposées par le collage
maPage.add(leConnecteur) ' ajouter avant de compléter
```

```
With leConnecteur
   .EdgeKind = com.sun.star.drawing.ConnectorType.LINES
   .StartShape = FindObjectByName(maPage, "carre1")
   .StartGluePointIndex = 1 ' le point à droite
   .EndShape = FindObjectByName(maPage, "cercle1")
   .EndGluePointIndex = 0 ' le point en haut
end With
End Sub
```

Pour voir les variantes possibles, changez la valeur de type de connecteur et les valeurs d'index. Pour un connecteur direct (`LINE`) utilisez directement la valeur 2, car la constante nommée n'est actuellement pas disponible.

Manipuler plusieurs formes

L'ordre Z

Une forme peut en cacher une autre. Tout dépend de l'ordre dans lequel elles sont affichées. Cet ordre est défini par la propriété `ZOrder`, de type `Long`, que possède chaque forme. Une forme peut cacher toutes celles qui possèdent un `ZOrder` inférieur au sien.

En lisant puis modifiant le `ZOrder` de vos formes, vous décidez de l'ordre effectif d'affichage, indépendamment de l'ordre dans lequel elles ont été insérées.

Grouper des formes

Pour grouper des formes qui se trouvent sur une même page, on les met dans le même sac, c'est-à-dire dans une collection de formes. Les formes sont ajoutées une à une dans la collection, puis celle-ci est donnée en argument à la fonction `group` de la page. La fonction renvoie un objet groupe. Le « dégroupage » utilise la méthode `ungroup` de la page. Elle prend comme argument l'objet groupe.

Notre exemple utilise des formes pré-existantes et nommées, pour alléger le code. Ouvrez l'EDI pour exécuter cette macro. Avant de répondre au message qui s'affichera, vous pourrez vérifier le groupage en déplaçant une des formes du groupe (cliquez dans le camemebert).

```
rem Code13-04.sxd   bibli : Selections Module3
Option Explicit

Sub GrouperFormes()
Dim monDocument As Object, maPage As Object
```

```
Dim uneForme As Object
Dim groupage As Object, monGroupe As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessins")

groupage = createUnoService(_
           "com.sun.star.drawing.ShapeCollection")
uneForme = FindObjectByName(maPage, "carreVide")
groupage.Add(uneForme)
uneForme = FindObjectByName(maPage, "cam")
groupage.Add(uneForme)
uneForme = FindObjectByName(maPage, "T01")
groupage.Add(uneForme)
monGroupe = maPage.group(groupage)
MsgBox("Cliquer pour dégrouper")
maPage.ungroup(monGroupe)
End Sub
```

Il est possible de déplacer ou redimensionner un groupe de formes, mais il n'est pas possible de modifier par macro les autres caractéristiques : trait, couleur, texte. Effectuez donc ces formatages avant de grouper les formes.

Combiner plusieurs formes

La combinaison de formes se réalise de manière quasi-identique à un groupement. On utilisera la fonction `combine` pour combiner, et la méthode `split` pour supprimer la combinaison. Comme avec l'interface utilisateur, combinaison et annulation de combinaison ne permettent pas de retrouver les objets originels.

```
rem Code13-04.sxd   bibli : Selections Module4
Sub CombinerFormes()
' - - - - - - - partie identique - - - - - -
monGroupe = maPage.combine(groupage)
MsgBox("Cliquer pour annuler la combinaison")
maPage.split(monGroupe)
' - - - - - - - partie identique - - - - - -
```

Connecter plusieurs formes

La connexion de formes se réalise de manière quasi-identique à un groupement. On utilisera la fonction `bind` pour connecter, et la méthode `unbind` pour supprimer la connexion. Comme avec l'interface utilisateur, connexion et annulation de connexion ne permettent pas de retrouver les objets originels.

```
rem Code13-04.sxd   bibli : Selections Module5
Sub ConnecterFormes()
' - - - - - - - partie identique - - - - - -
monGroupe = maPage.bind(groupage)
MsgBox("Cliquer pour annuler la connexion")
maPage.unbind(monGroupe)
' - - - - - - - partie identique - - - - - -
```

Exporter une forme

Exporter une forme consiste à en obtenir un fichier image codé dans un certain format. Le service d'exportation a pour nom :

```
com.sun.star.drawing.GraphicExportFilter
```

Il est capable de créer un fichier image à partir d'une page, d'une forme, ou d'une collection de formes. Il existe deux manières de préciser le codage du fichier image :

- avec un type Mime ; les types actuellement reconnus, avec l'extension usuelle des fichiers, sont listés au tableau 13-34,
- en utilisant les trois lettres d'extension spécifiques du codage, par exemple `wmf`.

Tableau 13–34 Types Mime reconnus

Mime	Format obtenu	Couleurs	Extension
image/x-MS-bmp	OS/2 - Windows bitmap	16 millions	bmp
application/postscript	Encapsulated Postscript	16 millions	eps
image/gif	Compuserve GIF 89a	256	gif
image/jpeg	JPEG version 1	16 millions	jpg, jpe, jpeg
image/png	Portable Network Graphic compressé	16 millions	png
image/tiff	Tagged Image File Format compressé	16 millions	tif, tiff
image/x-pict	Mac Pict	16	pct
image/x-portable-bitmap	Portable Bitmap	2	pbm
image/x-portable-graymap	Portable Graymap	256	
image/x-portable-pixmap	Portable Pixelmap	16 millions	ppm
image/x-cmu-raster	Sun Raster image, type 2	16 millions	ras
image/x-xpixmap	X Xindows Pixmap	256	xpm
image/svg+xml	Scalable Vector Graphics	16 millions	svg

Le mécanisme d'exportation ne documente pas les options de codage. Sans elles, vous ne pouvez pas choisir le taux de compression JPEG, ni choisir un codage 256 couleurs en PNG.

Le service d'exportation expose la liste exacte des types reconnus dans le tableau `SupportedMimeTypeNames` :

```
rem Code13-06.sxd   bibli : Exporter Module1
Option Explicit

Sub typesMimeExport()
Dim serv As Object, x As Long, liste As String
serv = CreateUnoService(_
       "com.sun.star.drawing.GraphicExportFilter")
liste = "- - Types Mime supportés - -"
for x = 0 to UBound(serv.SupportedMimeTypeNames())
  liste = liste & chr(13) & serv.SupportedMimeTypeNames(x)
next
MsgBox(liste)
End Sub
```

Pour exporter un objet, nous le mettons en argument de la méthode `setSourceDocument` de l'objet service. Notez que la pseudo-propriété OOoBasic `SourceDocument` ne fonctionne pas ici. Nous utilisons un tableau de deux propriétés pour indiquer :

- l'adresse du fichier à créer, sous forme d'une URL,
- le type Mime à utiliser.

Ce tableau de propriétés est fourni en argument à la méthode `filter` de l'objet service, qui effectue le travail.

```
rem Code13-06.sxd   bibli : Exporter Module2
Option Explicit

Sub exporterFormeMime()
Dim monDocument As Object, maPage As Object
Dim maForme As Object, serv As Object
Dim params(1) As New com.sun.star.beans.PropertyValue
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessins")
maForme = FindObjectByName(maPage, "ell1")
serv = CreateUnoService(_
       "com.sun.star.drawing.GraphicExportFilter")
serv.setSourceDocument(maForme) ' ne pas utiliser la propriété !
params(0).Name = "URL"
params(0).Value = convertToURL("C:\mafigure1.svg")
params(1).Name = "MediaType"
params(1).Value = "image/svg+xml"
serv.filter(params())
End Sub
```

Attention : l'exportation écrase tout fichier du même nom !

La deuxième méthode d'exportation est plus simple et peut permettre d'utiliser certains codages qui n'ont pas d'équivalent Mime (voir tableau 13-35).

Tableau 13–35 Codages d'image par l'extension

Extension	Signification	Couleurs
wmf	Windows Metafile	16 millions
emf	Windows Enhanced Metafile	16 millions
met	OS/2 Metafile	16 millions
svm	Starview Metafile	16 millions

Nous utilisons la propriété `FilterName`, de type `String`, pour préciser le filtre sous la forme de l'extension de fichier.

```
rem Code13-06.sxd   bibli : Exporter Module3
Option Explicit

Sub exporterFormeParExt()
Dim monDocument As Object, maPage As Object
Dim maForme As Object, serv As Object, filtre As String
Dim params(1) As New com.sun.star.beans.PropertyValue
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessins")
maForme = FindObjectByName(maPage, "rect2")
serv = CreateUnoService(_
       "com.sun.star.drawing.GraphicExportFilter")
serv.setSourceDocument(maForme)
filtre = InputBox("Extension du fichier, 3 lettres")
params(0).Name = "URL"
params(0).Value = convertToURL("C:\mafigure2." & filtre)
params(1).Name = "FilterName"
params(1).Value = filtre ' l'extension sert de nom de filtre
serv.filter(params())
End Sub
```

L'exportation de toute une page se ferait en donnant en argument de `setSourceDocument` l'objet page. Pour exporter un ensemble de formes, nous devons les mettre dans un objet collection, avec la méthode déjà vue avec le groupage des formes. Il est inutile de les grouper effectivement.

```
rem Code13-06.sxd   bibli : Exporter Module4
Option Explicit
```

```
Sub exporterDesFormes()
Dim monDocument As Object, maPage As Object
Dim uneForme As Object, serv As Object
Dim params(1) As New com.sun.star.beans.PropertyValue
Dim groupage As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Dessins")
groupage = createUnoService(_
           "com.sun.star.drawing.ShapeCollection")
uneForme = FindObjectByName(maPage, "ell1")
groupage.Add(uneForme)
uneForme = FindObjectByName(maPage, "rect1")
groupage.Add(uneForme)
uneForme = FindObjectByName(maPage, "conn1")
groupage.Add(uneForme)
serv = CreateUnoService("com.sun.star.drawing.GraphicExportFilter")
serv.setSourceDocument(groupage)
params(0).Name = "URL"
params(0).Value = convertToURL("C:\unGroupe.eps")
params(1).Name = "MediaType"
params(1).Value = "application/postscript"
serv.filter(params())
End Sub
```

Signalons que Sven Jacobi, de Sun Microsystems, a donné un exemple d'options pour l'exportation au format JPEG. On ajoute un paramètre supplémentaire :

```
params(2).Name = "FilterData"
params(2).Value = optionsFiltre()
```

La valeur `optionsFiltre` est un tableau de propriétés. En reprenant l'exemple de Sven cela donne :

```
Dim optionsFiltre(4) As New com.sun.star.beans.PropertyValue
optionsFiltre(0).Name  = "PixelWidth" ' largeur en pixels
optionsFiltre(0).Value = 1000
optionsFiltre(1).Name  = "PixelHeight" ' hauteur en pixels
optionsFiltre(1).Value = 1000
optionsFiltre(2).Name  ="LogicalWidth" ' largeur logique (?)
optionsFiltre(2).Value = 1000
optionsFiltre(3).Name  ="LogicalHeight" ' hauteur logique (?)
optionsFiltre(3).Value = 1000
optionsFiltre(4).Name  ="Quality" ' qualité souhaitée
optionsFiltre(4).Value = 60
```

Ce codage, non vérifié, est inspiré du document d'Andrew Pitonyak, version anglaise, dont nous parlons à l'annexe C.

Les styles

Il est de loin préférable de définir vos styles avec le styliste dans un document qui vous servira de modèle. Toutefois, vous pouvez avoir à faire quelques modifications d'un style existant, c'est ce que nous allons décrire.

Trouver les styles

Dans OpenOffice.org, les styles sont regroupés en familles. Les documents Draw ne contiennent qu'une seule famille de styles : `graphics`. Nous décrirons cependant un mécanisme plus général, qui nous sera utile pour Impress.

La macro suivante liste tous les styles d'un document Draw.

```
rem Code13-08.sxd   bibli : ModifStyles Module1
Option Explicit

Sub ListerStyles()
Dim monDocument As Object, nomFam As String
Dim lesFamilles As Object, uneFamille As Object
Dim styleX As Object, liste As String, cr As String
Dim f As Long, x As Long
cr = chr(13) ' retour à la ligne
monDocument = thisComponent

lesFamilles = monDocument.StyleFamilies
for f = 0 to lesFamilles.Count -1 ' chaque famille
  nomFam = lesFamilles.ElementNames(f)
  uneFamille = lesFamilles.getByName(nomFam)
  liste = "--- Famille : " & nomFam & cr
  for x = 0 to uneFamille.Count -1 ' chaque style dans famille
    styleX = uneFamille(x)
    liste = liste & styleX.Name & " = " & _
                    styleX.DisplayName & cr ' nom localisé
  next x
  MsgBox(liste)
next f
End Sub
```

La propriété `StyleFamilies` de l'objet document donne accès à toutes les familles de styles qui existent dans le document. Le nombre de familles est la propriété `Count` de l'objet `lesFamilles` ; la liste des noms de familles est disponible dans le tableau `ElementNames` de ce même objet.

Bien qu'avec OOoBasic nous puissions obtenir successivement chaque famille par une simple indexation de l'objet `lesFamilles`, cet ordre n'est pas forcément le même que dans `ElementNames`. C'est pourquoi nous obtenons chaque famille en utilisant la fonction `getByName` de l'objet `lesFamilles`.

Le nombre de styles dans une famille est indiqué par la propriété `Count` de celle-ci. Nous obtenons chaque style par une simple indexation. Le nom du style est obtenu avec sa propriété `Name`. Les styles prédéfinis en standard ont un nom interne anglais. Le nom localisé est récupéré grâce à la propriété `DisplayName`.

> **REMARQUE Noms internes des styles Draw**
>
> Si vous lisez attentivement les correspondances entre nom interne et nom localisé, vous observerez des bizarreries dans l'équivalent interne de `Titre`, `Titre 1`, `Titre 2`, `Titre 3`, `Titre 4`, `Titre 5`. Les trois premiers sont nommés `headline`, `headline1`, `headline2`, les trois autres sont nommés `title`, `title1`, `title2`. Cette nuance, qui est conservée sur les noms utilisateurs de la version anglaise, a été gommée par la traduction française.

Modifier un style existant

Cette macro récupère le style d'objet graphique prédéfini `Objet avec ombre` et change la couleur de remplissage. Avec le document du zip téléchargeable, affichez la page `Dessins`.

```
rem  Code13-08.sxd   bibli : ModifStyles Module2
Option Explicit

Sub ChangerStyle()
Dim monDocument As Object
Dim lesFamilles As Object, uneFamille As Object
Dim styleX As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("graphics")
styleX = uneFamille.getByName("Objet avec ombre")
styleX.FillColor = RGB(50,200,50) ' vert
End Sub
```

Attention à la casse ! Les noms de styles doivent être écrits en respectant les majuscules et minuscules.

En utilisant le Styliste vous constaterez que toutes les propriétés de formes sont réglables sur un style. Effectivement, vous retrouvez les propriétés que nous avons décrites (et d'autres encore). Il suffit de les modifier sur votre objet style pour changer celui-ci.

Supprimer un style existant

Pour supprimer un style personnel existant, on utilise la méthode `removeByName` :

```
uneFamille.removeByName("monStyle")
```

On ne doit pas supprimer un style prédéfini.

Créer un nouveau style

Nous devons d'abord insérer un nouveau style dans sa famille, puis le modifier. La macro ci-dessous crée un nouveau style d'objet `Objet ombre à gauche`, l'initialise à partir du style existant `Objet avec ombre` et change une de ses propriétés. Fermez puis rouvrez le Styliste pour faire apparaître le nouveau style.

```
rem  Code13-08.sxd   bibli : ModifStyles Module3
Option Explicit

Sub HeriterStyle()
Dim monDocument As Object
Dim lesFamilles As Object, uneFamille As Object
Dim nouvStyle As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("graphics")
nouvStyle = monDocument.CreateInstance(_
                        "com.sun.star.style.Style")
uneFamille.insertByName("Objet ombre à gauche", nouvStyle)
nouvStyle.ParentStyle = "Objet avec ombre" ' hériter d'un style
nouvStyle.ShadowXDistance = -300
End Sub
```

Imprimer

Le mécanisme général d'impression est décrit au chapitre 10. Nous traiterons ici des particularités de Draw.

Chaque forme dispose d'une propriété `Printable`, de type `Boolean`. La valeur `True` autorise son impression. De même, chaque couche expose une propriété `IsPrintable` qui permet d'imprimer ou non les objets qui lui sont associés.

Configuration d'impression

Les propriétés listées au tableau 13-36 correspondent aux cases à cocher de l'interface utilisateur Impression > Options... Il s'agit de propriétés de configuration du document, qui sont obtenues en invoquant le service `DocumentSettings`, comme dans cet exemple.

```
rem Code10-03.sxd   bibli : Imprimer Module4
Option Explicit

Sub ConfigImpression()
Dim monDocument As Object, conf As Object, servConfig As String
```

```
monDocument = ThisComponent
servConfig = "com.sun.star.drawing.DocumentSettings"
conf = monDocument.createInstance(servConfig)
print conf.PrinterName
End Sub
```

Tableau 13–36 Propriétés d'impression pour Draw

Propriété	Type	Signification
IsPrintFitPage	Boolean	True pour adapter la page à l'espace imprimable.
IsPrintTilePage	Boolean	True pour imprimer en mosaïque.
PrinterName	String	Nom de l'imprimante utilisée par le document.
IsPrintPageName	Boolean	True pour imprimer le nom de la page.
IsPrintDate	Boolean	True pour imprimer la date.
IsPrintTime	Boolean	True pour imprimer l'heure.
IsPrintBooklet	Boolean	True pour imprimer en format prospectus.
IsPrintBookletBack	Boolean	True pour n'imprimer que le verso du prospectus.
IsPrintBookletFront	Boolean	True pour n'imprimer que le recto du prospectus.
PrintQuality	Long	Qualité d'impression 0 : Normal 1 : Niveaux de gris 2 : Noir et Blanc seulement

Configuration du document

Les propriétés listées au tableau 13-37 concernent la mise à l'échelle du document et l'unité de mesure employée (pour les cotations). Avec l'interface utilisateur, ceci est défini dans le menu Outils > Options > Dessin > Général. Par programmation, on accède à ces propriétés en invoquant le service DocumentSettings pour le document en cours.

Tableau 13–37 Propriétés de document Draw

Propriété	Type	Signification
MeasureUnit	Integer	Unité de mesure par défaut, constante nommée.
ScaleNumerator	Long	Numérateur de l'échelle du document.
ScaleDenominator	Long	Dénominateur de l'échelle du document.

Les principales unités de mesure sont listées au tableau 13-38. Les constantes nommées sont de la forme :

```
com.sun.star.util.MeasureUnit.MM
```

Tableau 13–38 Constantes d'unité de mesure

Constante	Signification
MM_100TH	1/100 de mm.
MM	mm
CM	cm
M	m
KM	Km
TWIP	Twip
INCH	Pouce
FOOT	Pied
MILE	Mile
PICA	Pica
POINT	Point
PERCENT	Pourcentage

L'échelle ne donne des résultats cohérents que pour un rapport Numérateur/Dénominateur inférieur ou égal à 1. Dans l'exemple suivant, nous avons placé des cotations sur la page de dessin. La macro va changer l'échelle de cotation pour le document. Observez le changement des valeurs affichées sur le dessin.

```
rem Code13-09.sxd   bibli : Config Module1
Option Explicit

Sub ConfigEchelle()
Dim monDocument As Object, conf As Object, servConfig As String
monDocument = ThisComponent
servConfig = "com.sun.star.drawing.DocumentSettings"
conf = monDocument.createInstance(servConfig)
' alterner entre deux échelles très différentes !
if conf.MeasureUnit = com.sun.star.util.MeasureUnit.MM then
  conf.MeasureUnit = com.sun.star.util.MeasureUnit.KM
  conf.ScaleNumerator = 3
  conf.ScaleDenominator = 100000
else
  conf.MeasureUnit = com.sun.star.util.MeasureUnit.MM
  conf.ScaleNumerator = 7
  conf.ScaleDenominator = 10
end if
End Sub
```

Spécificités d'Impress par rapport à Draw

Nous décrivons ici les quelques particularités d'Impress.

Couches de dessin (version 2.0.0)

Les couches de dessin ne sont plus disponibles dans Impress version 2.0.0. Suite à la demande des utilisateurs, elles reviendront dans la future version OpenOffice.org 2.0.1.

Exécuter une macro pendant une présentation

Dans OpenOffice.org 1.1, il est possible de lancer une macro par un simple clic sur un objet d'une diapo, pendant la présentation. Pour cela, en mode dessin, sélectionnez l'objet. Dans le menu contextuel, choisissez Interaction > Action par clic de souris > Exécuter la macro. Recherchez le nom de la macro avec le bouton Parcourir.

> **LIMITATION** **Impress version 2.0**
>
> Suite à des choix de développement, à partir de la version 2.0 d'OpenOffice.org il n'est plus possible de modifier, par action directe de l'API, l'affichage d'une diapo pendant l'exécution de la présentation. La macro fonctionnera correctement en mode conception, mais ne fera aucun changement visible durant la présentation. La version 2.0 apporte de nombreuses animations et transitions, aussi le recours à une macro pour un effet spécial est-il moins utile que dans la version précédente.

La page de notes

Chaque page de dessin d'un document Impress est associée à une page de note, qu'on peut obtenir avec la propriété `NotesPage` de la page. Cette page de notes a toutes les propriétés d'une page de dessin : tout ce qui a été dit à ce propos lui est applicable. Par défaut, on trouve dans la page de notes deux formes, chacune d'un type spécial :

```
com.sun.star.presentation.PageShape
com.sun.star.presentation.NotesShape
```

La première contient une reproduction de la diapo, la deuxième contient les notes proprement dites. Notez (!) qu'une page de notes a elle aussi une propriété `NotesPage` ! Elle n'est d'aucune utilité, car elle donne seulement accès à elle-même.

Nous allons énumérer les objets qui se trouvent sur la page de notes de la page `Diapo1` et afficher le texte de la note. Cette macro est très similaire à l'énumération des formes dans une page de dessin, vue plus haut.

```
rem Code10-03.sxd   bibli : Library1 Module1
Option Explicit

Sub formesDeNotes()
Dim monDocument As Object, maPage As Object, maNote As Object
Dim maForme As Object, typF As String, n As Long
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Diapo1")
maNote = maPage.NotesPage
for n = 0 to maNote.Count -1
  maForme = maNote(n)
  typF = maForme.ShapeType
  print "Type de la forme = " & typF
  if  typF = "com.sun.star.presentation.NotesShape" then
    MsgBox(maForme.Text.String) ' texte de la note
  end if
next
End Sub
```

Chaque arrière-plan est aussi associé à sa page de notes ; on accède aussi à cette dernière par la propriété `NotesPage` de l'objet arrière-plan.

La page prospectus

Le mode d'affichage prospectus (en anglais *handout*) reproduit plusieurs diapos sur une même page, pour fournir un résumé imprimé de la présentation. Le terme français est mal choisi, et de plus identique au mode d'impression prospectus qui sert effectivement à imprimer de petites brochures ou prospectus.

L'objet page de prospectus est accessible par la propriété `HandoutMasterPage` de l'objet document. Il y a peu à dire de cette page ; on ne peut modifier avec l'API le nombre de diapos qu'elle contient, seulement le connaître avec la propriété `Count`.

Les styles Impress

Alors que Draw utilise une seule famille de styles : `graphics`, Impress utilise deux familles : `graphics` et `Standard`. Vous pouvez le vérifier en exécutant la macro `ListerStyles()` dans un document Impress (voir zip téléchargeable, `Code13-07.sxi`). La famille `Standard` correspond aux Styles d'objets de présentation dans le Styliste.

Le listage des styles réserve une surprise avec la famille `Standard` : les noms localisés sont en *allemand* ! (constaté sur OpenOffice.org 1.1.1 et antérieure, version française et version anglaise).

Deuxième problème, lié au premier, les fonctions `hasByName` et `getByName` ne reconnaissent que le nom interne *anglais* pour les styles de la famille `Standard`. Voici la correspondance nom interne / nom localisé :

- `title` : Titre
- `subtitle` : Sous-titre
- `background` : Arrière-plan
- `backgroundobjects` : Objets d'arrière-plan
- `notes` : Notes
- `outline1` à `outline9` : Plan 1 à Plan 9.

Nous ne pouvons pas créer de nouveaux styles dans la famille `Standard`, seulement les modifier. Nous ne pouvons pas non plus supprimer un de ces styles. Les propriétés ont pour la plupart déjà été décrites. Cet exemple impose une couleur de fond uniforme au style `Arrière-plan`.

```
rem Code13-07.sxi   bibli : ModifStyles Module2
Option Explicit

Sub ModifierStyleImpress()
Dim monDocument As Object, leStyle As Object
Dim lesFamilles As Object, uneFamille As Object
monDocument = ThisComponent
lesFamilles = monDocument.StyleFamilies
uneFamille = lesFamilles.getByName("Standard")
leStyle = uneFamille.getByName("background")
leStyle.FillColor = RGB(100,255,255)
leStyle.FillStyle = com.sun.star.drawing.FillStyle.SOLID
End Sub
```

Configuration de document

Les propriétés d'unité de mesure et d'échelle ne sont pas disponibles.

Configuration d'impression

Pour obtenir la configuration d'impression d'un document Impress, on emploiera le service :

```
servConfig = "com.sun.star.presentation.DocumentSettings"
```

On obtient les propriétés d'impressions décrites pour Draw, et quelques autres listées au tableau 13-39, spécifiques à Impress.

Tableau 13–39 Propriétés d'impression spécifiques pour Impress

Propriété	Type	Signification
`IsPrintDrawing`	`Boolean`	`True` pour imprimer les diapos.
`IsPrintNotes`	`Boolean`	`True` pour imprimer les notes de page.
`IsPrintHandout`	`Boolean`	`True` pour imprimer les diapos à plusieurs par page (résumé de présentation).
`IsPrintOutline`	`Boolean`	`True` pour imprimer le plan.
`IsPrintHiddenPages`	`Boolean`	`True` pour imprimer les diapos cachées.

Accès aux mécanismes d'Impress

L'API donne accès à une partie des mécanismes propres à une présentation Impress. Cependant, nous ne les décrirons pas car il nous a semblé peu utile de réaliser par macro des transitions de diapo ou d'objets, ou des présentations personnalisées.

> Ces aspects sont décrits dans le « Developer's Guide », chapitre 9.5.2, qui donne de nombreux liens vers le SDK.

Conclusion

De nombreux objets et méthodes à vocation graphique, tels que les primitives de bases mais aussi les points de colle ou les groupements, sont disponibles. Un document Impress est avant tout un document Draw enrichi de fonctionnalités supplémentaires que nous avons détaillées.

Le chapitre suivant, transversal, va aborder l'insertion d'objets tels que les images et équations dans les documents OpenOffice.org.

14

Les objets insérables dans un document

Différents types d'objets peuvent être insérés dans les documents OpenOffice.org (Writer, Calc, Draw, Impress). Vous en avez déjà rencontré certains aux chapitres précédents. Nous allons à présent les passer en revue et décrire les aspects spécifiques à la gestion des formes et des images.

> Afin que chaque partie puisse être lue indépendamment des autres, nous n'avons pas hésité à introduire une certaine redondance.

Classification des objets insérables

Ces objets sont décomposables en types et sous-types, selon la figure 14-1. Tous ces objets peuvent être insérés dans un document texte, et la plupart dans les autres documents OpenOffice.org.

Les objets regroupés sous l'appellation `BaseFrame` sont disponibles dans les collections :

- `TextFrames`,
- `TextEmbeddedObjects`,
- `TextGraphicObjects`.

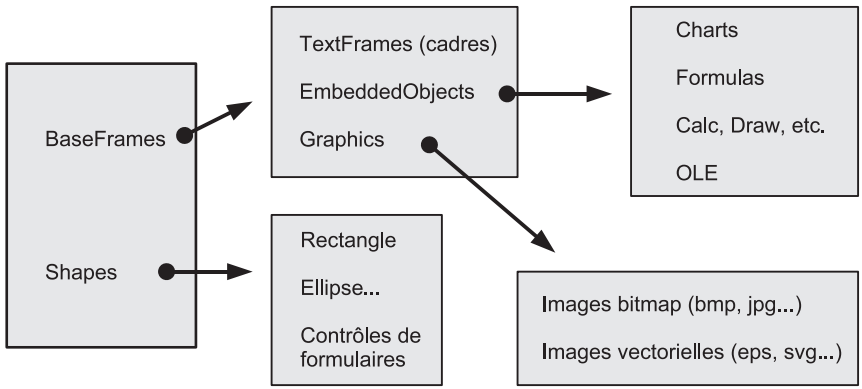


Figure 14–1
Classification des objets insérables

Chacune de ces collections est capable de retrouver un de ses objets par son nom (`getByName`, `hasByName`), ou par indexation.

Notez qu'il n'est pas possible de créer ou insérer d'`embeddedObjects` en l'état actuel de l'API. Seule l'intervention sur des objets existants est possible.

> Les objets intégrés font l'objet d'une discussion technique au chapitre 7.3.10 du « Developer's Guide » (en anglais) du SDK.

Les formes, ou dessins

La gestion des formes, ou dessins, est décrite en détail dans le chapitre 13 consacré aux documents Draw, dans les sections expliquant l'insertion de formes et leurs propriétés. Si vous n'êtes intéressé que par Writer ou Calc, vous trouverez ici les aspects qui leur sont spécifiques, après quoi vous pourrez poursuivre votre lecture au chapitre 13.

Les formes dans un document Writer

Les formes d'un document Writer sont placées sur une unique page de dessin (en anglais *draw page*), qui s'étend sur le document tout entier. L'objet page est accessible ainsi :

```
Dim maPage As Object
maPage = monDocument.DrawPage
```

Après avoir obtenu un objet forme, son insertion sur la page de dessin se fait facilement :

```
maPage.add(maForme)
```

Malheureusement, le positionnement est alors obligatoirement ancré à la page de dessin, ce qui pose problème avec un document de plusieurs pages de texte, aussi emploierons-nous une autre méthode.

Cette page de dessin dans Writer ne comporte pas les propriétés d'une page Draw, mais uniquement la liste des objets sur cette page. Et à la différence d'un document Draw, un document Writer ne comporte ni arrière-plan ni couche.

Insérer une forme à la position du curseur

La meilleure solution pour insérer une forme est d'utiliser un curseur d'écriture et la méthode `insertTextContent` de l'objet texte. Nous allons insérer une ellipse. Le principe ressemble pour partie à l'insertion d'un cadre dans un document Writer, et pour partie à l'insertion d'une forme dans Draw.

```
Rem Code14-01.sxw   bibli : Dessins Module1
Option Explicit

Sub AjouterEllipse()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, maForme As Object
Dim dimensionsForme As New com.sun.star.awt.Size
Dim positionForme As New com.sun.star.awt.Point
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False) ' déplacer le curseur

maForme = monDocument.createInstance("com.sun.star.drawing.EllipseShape")
dimensionsForme.Width = 2600 ' 26 mm de large
dimensionsForme.Height = 1200 ' 12 mm de haut
positionForme.x = 3500 ' 35 mm à droite du point d'ancrage
positionForme.y = 1300 ' 13 mm en dessous du point d'ancrage
maForme.Size = dimensionsForme
maForme.AnchorType = _
        com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
monTexte.insertTextContent(monCurseur, maForme, false)
maForme.Position = positionForme
maForme.Surround = com.sun.star.text.WrapTextMode.RIGHT
End Sub
```

Pour les formes, contrairement aux cadres, seul le positionnement absolu est possible. Les distances sont exprimées en 1/100 de millimètres, mesurées du coin haut-gauche de l'ancre au coin haut-gauche de la forme. Les valeurs possibles d'ancrage sont celles listées pour ancrer un cadre.

Une forme dispose de toutes les propriétés d'adaptation du texte déjà vues à propos des cadres, auxquelles s'ajoutent deux autres propriétés :

- `ContourOutside`, de type `Boolean`, sert à éviter que le texte apparaisse dans la partie concave d'une figure complexe.
- `SurroundContour`, de type `Boolean`, correspond à l'adaptation `Contour`.

Insérer plusieurs formes

À chaque insertion d'une forme, il est nécessaire d'obtenir un nouvel objet forme, même si on insère plusieurs fois le même type de forme. Il faut aussi réinitialiser à chaque fois les propriétés de la forme. Dans cet exemple, on insère deux rectangles identiques à la même position, ancrés à la page, l'un dans la page 4, l'autre dans la page 2 du document Writer. Nous leur donnons un nom, qui nous servira dans le prochain exemple.

```
Rem Code14-01.sxw   bibli : Dessins Module2
Option Explicit

Sub PlusieursFormes()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, maForme As Object
Dim dimensionsForme As New com.sun.star.awt.Size
Dim positionForme As New com.sun.star.awt.Point
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor

' pour un ancrage page la position du curseur est indifférente
dimensionsForme.Width = 8400 ' 84 mm de large
dimensionsForme.Height = 2530 ' 25,3 mm de haut
positionForme.x = 3500 ' 35 mm à droite du point d'ancrage
positionForme.y = 5300 ' 53 mm en dessous du point d'ancrage

' première forme insérée sur la page 4
maForme = monDocument.createInstance("com.sun.star.drawing.RectangleShape")
maForme.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
maForme.AnchorPageNo = 4
monTexte.insertTextContent(monCurseur, maForme, false)
maForme.Size = dimensionsForme
maForme.Position = positionForme
maForme.Name = "Rect1" ' donner un nom à cette forme

' deuxième forme en page 2, position et dimensions identiques
maForme = monDocument.createInstance("com.sun.star.drawing.RectangleShape")
maForme.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
maForme.AnchorPageNo = 2
monTexte.insertTextContent(monCurseur, maForme, false)
maForme.Size = dimensionsForme
maForme.Position = positionForme
maForme.Name = "Rect2" ' donner un nom à cette forme
End Sub
```

Retrouver, supprimer une forme

Le Navigateur de Writer n'est pas capable de lister les objets dessins d'un document. Néanmoins, par macro nous somme capables de les retrouver sur la page de dessin. De manière très similaire à Draw, nous allons sélectionner une des formes du document.

```
Rem Code14-01.sxw   bibli : Dessins Module3
Option Explicit

Sub SelectionnerForme()
Dim monDocument As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
maPage = monDocument.DrawPage
maForme = FindObjectByName(maPage, "Rect1")
if IsNull(maForme) then
  print "Il n'existe aucune forme de ce nom"
else
  monDocument.CurrentController.Select(maForme)
end if
End Sub
```

Comme dans Draw, il est possible de supprimer une forme à partir de la page de dessin.

```
maPage.remove(maForme)
```

Interaction entre la forme et les autres objets

Une forme insérée avec la méthode `insertTextContent` de l'objet texte du document se place par défaut en arrière-plan par rapport à un cadre, un tableau, un en-tête ou un bas de page. La forme pourra cependant déplacer un élément de tableau, en fonction de l'adaptation au texte.

L'insertion de forme par la méthode `insertTextContent` de l'objet texte fonctionne aussi avec un objet texte obtenu d'un cadre, d'un en-tête ou d'un bas de page. Cependant, la forme se trouve alors aussi en arrière-plan, masquée par l'objet auquel elle est ancrée.

On fait passer la forme au premier-plan en affectant la valeur `True` à sa propriété `Opaque`, de type `Boolean`. La valeur `False` la renvoie à l'arrière-plan.

```
maForme.Opaque = true ' passer la forme au premier plan
```

Notez que le fait d'ancrer une forme à un cadre, par exemple, ne la contraint pas à rester dans ce cadre. Elle peut en déborder ou se trouver en dehors.

Les formes dans un document Calc

Chaque feuille d'un document Calc dispose d'une page de dessin. L'objet page est accessible ainsi :

```
Dim maPage As Object
maPage = maFeuille.DrawPage
```

L'insertion d'une forme sur la page de dessin se fait en utilisant la propriété `add` de la page de dessin.

```
Rem Code14-02.sxc   bibli : Dessins Module1
Option Explicit

Sub AjouterEllipse()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maPage As Object, maForme As Object
Dim dimensionsForme As New com.sun.star.awt.Size
Dim positionForme As New com.sun.star.awt.Point
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("F01")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
dimensionsForme.Width = 2600 ' 26 mm de large
dimensionsForme.Height = 1200 ' 12 mm de haut
positionForme.x = 3500 ' 35 mm à droite du coin de la feuille
positionForme.y = 3300 ' 33 mm en dessous du coin de la feuille

maForme = monDocument.createInstance("com.sun.star.drawing.EllipseShape")
maForme.Size = dimensionsForme
maPage.add(maForme)
maForme.Position = positionForme
End Sub
```

Si vous sélectionnez manuellement la forme obtenue sur la feuille, vous verrez qu'elle est ancrée à une cellule de la feuille, par exemple B8. L'API a choisi comme cellule d'ancrage celle qui englobe le point de positionnement, et a complété le positionnement de la forme par rapport à cette cellule pour obtenir la position voulue. Effacez la forme ; modifiez la largeur de la colonne A et la hauteur d'une des premières lignes ; ré-exécutez la macro : le choix de la cellule d'ancrage changera. Ceci signifie que le positionnement est toujours calculé *par rapport à la page* et converti en positionnement à la cellule.

Si, après insertion de la forme, on modifie la largeur d'une colonne ou la hauteur d'une ligne recouverte par la forme, ses dimensions changeront. Si on modifie la largeur d'une colonne précédente ou la hauteur d'une ligne précédente, la position de la forme changera.

Parmi les propriétés de forme, décrites au chapitre 13, la propriété `Opaque` permet de mettre la forme en arrière-plan, c'est-à-dire derrière les cellules du tableur.

Insérer plusieurs formes

À chaque insertion d'une forme, il est nécessaire d'obtenir un nouvel objet forme, même si on insère plusieurs fois le même type de forme. Il faut aussi réinitialiser à chaque fois les propriétés de la forme. Dans cet exemple, on insère deux rectangles identiques à la même position, ancrés sur deux feuilles différentes du document Calc. Nous leur donnons un nom, qui nous servira dans le prochain exemple.

```
Rem Code14-02.sxc   bibli : Dessins Module2
Option Explicit

Sub PlusieursFormes()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maPage As Object, maForme As Object
Dim dimensionsForme As New com.sun.star.awt.Size
Dim positionForme As New com.sun.star.awt.Point

monDocument = thisComponent
lesFeuilles = monDocument.Sheets
dimensionsForme.Width = 5500 ' 55 mm de large
dimensionsForme.Height = 1200 ' 12 mm de haut
positionForme.x = 3500 ' 35 mm à droite du coin de la feuille
positionForme.y = 7200 ' 72 mm en dessous du coin de la feuille

' première forme insérée sur la feuille F01
maFeuille = lesFeuilles.getByName("F01")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
maForme = monDocument.createInstance("com.sun.star.drawing.RectangleShape")
maForme.Size = dimensionsForme
maPage.add(maForme)
maForme.Position = positionForme
maForme.Name = "Rect1" ' donner un nom à cette forme

' deuxième forme insérée sur la feuille F02
maFeuille = lesFeuilles.getByName("F02")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
maForme = monDocument.createInstance("com.sun.star.drawing.RectangleShape")
maForme.Size = dimensionsForme
maPage.add(maForme)
maForme.Position = positionForme
maForme.Name = "Rect2" ' donner un nom à cette forme
End Sub
```

Retrouver, supprimer une forme

Le Navigateur de Calc n'est pas capable de lister les objets dessins d'un document. Néanmoins, par macro nous sommes capables de les retrouver sur la page de dessin d'une feuille. De manière très similaire à Draw, nous allons sélectionner une des formes du document.

```
Rem Code14-02.sxc   bibli : Dessins Module3
Option Explicit

Sub SelectionnerForme()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maPage As Object
Dim maForme As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("F01")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
maForme = FindObjectByName(maPage, "Rect1")
if IsNull(maForme) then
  print "Il n'existe aucune forme de ce nom"
else
  monDocument.CurrentController.Select(maForme)
end if
End Sub
```

Comme dans Draw, il est possible de supprimer une forme à partir de la page de dessin.

```
maPage.remove(maForme)
```

Les images

Nous allons voir comment insérer des images dans un document OpenOffice.org. Nous attirons votre attention sur le fait que seul un *lien* vers l'image est inséré et non l'image elle-même. Ceci est une limitation actuelle de l'API. Si vous déplacez, renommez, supprimez le fichier image référencé, le document affichera une case de lien brisé avec l'adresse URL du fichier. Pour effectivement insérer dans le document des images liées, suivre cette procédure manuelle :

1. ouvrir le menu Édition > Liens ;
2. sélectionner l'ensemble des liens images ;
3. cliquer sur le bouton Déconnecter ; confirmer ;
4. sauver le document.

BOGUE Suppression du lien

Quand un lien d'image est supprimé, OpenOffice.org insère dans le document le fichier image au format PNG. Or, ce format comprime très peu le fichier en comparaison du format JPEG. Le document final peut alors grossir énormément s'il contient des images photographiques. La correction de ce défaut est prévue pour la version 2.0.

Avant d'aborder les aspects logiciels, quelques notions sur les images sont nécessaires. Une image informatique *bitmap* ne possède pas de dimensions physiques par elle-même. Elle est caractérisée par le nombre de points élémentaires (pixels) qui la composent. Ces points (colorés et de forme carrée) sont disposés en matrice rectangulaire et on parle d'une largeur et d'une hauteur en nombre de pixels. Une image est reproduite sur un écran ou une feuille de papier en effectuant une projection, qui affecte une certaine taille physique à chaque pixel, ou, ce qui revient au même, en décidant du nombre de pixels par unité de mesure physique. On emploie habituellement le nombre de points par pouce ou PPP (un pouce = 25,4 mm), désigné en anglais par le signe DPI (*Dots Per Inch*). Une imprimante laser offre une densité typique de 300 ou 600 DPI.

Pour dimensionner une image sur un document, nous aurons besoin d'effectuer une « projection » des pixels, en veillant à conserver les proportions de l'image. Pour cela, nous utiliserons la routine utilitaire `resizeImageByWidth`, que nous avons extraite de l'annexe B de ce livre. Cette routine est recopiée dans la bibliothèque `Standard` de chaque document exemple du zip téléchargeable.

Les images dans Draw

L'insertion (du lien) d'une image dans Draw offre des similarités avec l'insertion d'une forme.

```
Rem Code14-03.sxd   bibli : lesImages Module1
Option Explicit

Sub AjouterImage()
Dim monDocument As Object, maPage As Object
Dim monImage As Object
Dim positionImage As New com.sun.star.awt.Point
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Photos1")

positionImage.x = 6500 ' 65 mm à droite du coin de la page
positionImage.y = 5300 ' 53 mm en dessous du coin de la page

monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
maPage.add(monImage)
resizeImageByWidth(monImage, 11000) ' largeur en 1/100 de mm
monImage.Position = positionImage
End Sub
```

Nous insérons une forme de type `GraphicObjectShape`. La propriété `GraphicURL` reçoit le chemin complet du fichier contenant l'image. L'image est dimensionnée *après* son insertion, en fixant sa largeur. Sans la macro `resizeImageByWidth`, nous pourrions imposer directement largeur et hauteur, mais au risque de déformer l'image. Le positionnement de l'image n'offre rien de nouveau.

Dans le jargon de l'API, le terme anglais *Graphic* est utilisé pour désigner une image *bitmap* ou vectorielle. Aucun rapport donc avec le concept français de graphique.

Insérer plusieurs images

Pour insérer plusieurs images, il est nécessaire d'obtenir à chaque fois un nouvel objet image grâce à `createInstance`, même pour plusieurs images identiques. Il est possible de nommer une image en utilisant sa propriété `Name`.

```
Rem Code14-03.sxd   bibli : lesImages Module2
Option Explicit

Sub PlusieursImages()
Dim monDocument As Object, maPage As Object
Dim monImage As Object
Dim positionImage As New com.sun.star.awt.Point

monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Photos1")
' première image
positionImage.x = 6500 ' 65 mm à droite du coin de la page
positionImage.y = 5300 ' 53 mm en dessous du coin de la page

monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
maPage.add(monImage)
resizeImageByWidth(monImage, 11000) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Name = "Logo1"
' deuxième image
positionImage.x = 700
positionImage.y = 14100

monImage = monDocument.createInstance(_"com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
maPage.add(monImage)
resizeImageByWidth(monImage, 5700) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Name = "Logo2"
End Sub
```

Retrouver, supprimer une image

L'image ayant été nommée, elle peut être retrouvée sur une page de dessin en utilisant la routine utilitaire de l'annexe B : `FindObjectByName`.

```
Rem Code14-03.sxd   bibli : lesImages Module3
Option Explicit

Sub SelectionnerImage()
Dim monDocument As Object, maPage As Object
Dim monImage As Object
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Photos1")
monImage = FindObjectByName(maPage, "Logo2")
if IsNull(monImage) then
  print "Il n'existe aucune image de ce nom"
else
  monDocument.CurrentController.Select(monImage)
end if
End Sub
```

La méthode `remove` de l'objet page de dessin permet de supprimer une image :

```
maPage.remove(monImage)
```

Les images dans un document Writer

Les images d'un document Writer sont placées sur une unique page de dessin (en anglais *draw page*), qui s'étend sur le document tout entier. L'objet page est accessible ainsi :

```
Dim maPage As Object
maPage = monDocument.DrawPage
```

Après avoir obtenu un objet image, son insertion sur la page de dessin se fait facilement :

```
maPage.add(monImage)
```

Malheureusement, le positionnement est alors obligatoirement ancré à la page de dessin, ce qui pose problème avec un document de plusieurs pages de texte ; aussi emploierons-nous une autre méthode.

> **RAPPEL**
>
> La page de dessin Writer ne comporte pas les propriétés d'une page Draw, mais seulement la liste des objets sur cette page. À la différence de Draw, un document Writer ne comporte ni arrière-plan ni couche.

Insérer une image à la position du curseur

La meilleure solution pour insérer une forme est d'utiliser un curseur d'écriture et la méthode `insertTextContent` de l'objet texte. Le principe ressemble pour partie à l'insertion d'un cadre dans un document Writer, et pour partie à l'insertion d'une forme dans Draw.

```
Rem Code14-01.sxw   bibli : lesImages Module1
Option Explicit

Sub AjouterImage()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, monImage As Object
Dim positionImage As New com.sun.star.awt.Point
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
monCurseur.gotoNextParagraph(False) ' déplacer le curseur
positionImage.x = 1500 ' 15 mm à droite du point d'ancrage
positionImage.y = 1300 ' 13 mm en dessous du point d'ancrage

monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
monImage.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
monTexte.insertTextContent(monCurseur, monImage, false)
resizeImageByWidth(monImage, 5500) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Surround = com.sun.star.text.WrapTextMode.RIGHT
End Sub
```

Pour les images, contrairement aux cadres, seul le positionnement absolu est possible. Les distances sont exprimées en 1/100 de millimètres, mesurées du coin haut-gauche de l'ancre au coin haut-gauche de la forme. Les valeurs possibles d'ancrage sont celles listées pour ancrer un cadre. Le Navigateur est capable de lister les images du document.

Insérer plusieurs images

Pour insérer plusieurs images, il est nécessaire d'obtenir à chaque fois un nouvel objet image grâce à `createInstance`, même pour plusieurs images identiques. Dans cet exemple, on insère deux images identiques à la même position, ancrées à la page, l'une dans la page 4, l'autre dans la page 2 du document Writer. Il est possible de nommer une image en utilisant sa propriété `Name`.

```
Rem Code14-01.sxw   bibli : Dessins Module2
Option Explicit

Sub PlusieursImages()
Dim monDocument As Object, monTexte As Object
Dim monCurseur As Object, monImage As Object
Dim positionImage As New com.sun.star.awt.Point
monDocument = ThisComponent
monTexte = monDocument.Text
monCurseur = monTexte.createTextCursor
```

```
' pour un ancrage page la position du curseur est indifférente
positionImage.x = 3500 ' 35 mm à droite du point d'ancrage
positionImage.y = 5300 ' 53 mm en dessous du point d'ancrage

' première image insérée sur la page 4
monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
monImage.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
monImage.AnchorPageNo = 4
monTexte.insertTextContent(monCurseur, monImage, false)
resizeImageByWidth(monImage, 5500) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Name = "Logo1" ' donner un nom à cette image

' deuxième forme en page 2, position et dimensions identiques
monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
monImage.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
monImage.AnchorPageNo = 2
monTexte.insertTextContent(monCurseur, monImage, false)
resizeImageByWidth(monImage, 5500) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Name = "Logo2" ' donner un nom à cette image
End Sub
```

Retrouver, supprimer une image

L'image ayant été nommée, elle peut être retrouvée sur la page de dessin en utilisant la routine utilitaire de l'annexe B : `FindObjectByName`. Puis, nous sélectionnerons cette image.

```
Rem Code14-01.sxw   bibli : Dessins Module3
Option Explicit

Sub SelectionnerImage()
Dim monDocument As Object, maPage As Object
Dim monImage As Object
monDocument = thisComponent
maPage = monDocument.DrawPage
monImage = FindObjectByName(maPage, "Logo1")
if IsNull(monImage) then
  print "Il n'existe aucune image de ce nom"
else
  monDocument.CurrentController.Select(monImage)
end if
End Sub
```

La méthode `remove` de l'objet page de dessin permet de supprimer une image :

```
maPage.remove(monImage)
```

Les images dans un document Calc

Chaque feuille d'un document Calc dispose d'une page de dessin. L'objet page est accessible ainsi :

```
Dim maPage As Object
maPage = maFeuille.DrawPage
```

L'insertion d'une image sur la page de dessin se fait en utilisant la propriété `add` de la page de dessin.

```
Rem Code14-02.sxc   bibli : lesImages Module1
Option Explicit

Sub AjouterImage()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maPage As Object, monImage As Object
Dim positionImage As New com.sun.star.awt.Point
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("F01")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
positionImage.x = 3500 ' 35 mm à droite du coin de la feuille
positionImage.y = 3300 ' 33 mm en dessous du coin de la feuille

monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
maPage.add(monImage)
resizeImageByWidth(monImage, 5500) ' largeur en 1/100 de mm
monImage.Position = positionImage
End Sub
```

Le Navigateur est capable de lister les images du document.

Insérer plusieurs images

Pour insérer plusieurs images, il est nécessaire d'obtenir à chaque fois un nouvel objet image grâce à `createInstance`, même pour plusieurs images identiques. Dans cet exemple, on insère deux images identiques à la même position, ancrées sur deux feuilles différentes du document Calc. Il est possible de nommer une image en utilisant sa propriété `Name`.

```
Rem Code14-02.sxc   bibli : lesImages Module2
Option Explicit

Sub PlusieursImages()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maPage As Object, monImage As Object
Dim positionImage As New com.sun.star.awt.Point
```

```
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
positionImage.x = 3500 ' 35 mm à droite du coin de la feuille
positionImage.y = 7200 ' 72 mm en dessous du coin de la feuille

' première image insérée sur la feuille F01
maFeuille = lesFeuilles.getByName("F01")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
maPage.add(monImage)
resizeImageByWidth(monImage, 11000) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Name = "Logo1" ' donner un nom à cette image

' deuxième image insérée sur la feuille F02
maFeuille = lesFeuilles.getByName("F02")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
monImage = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicURL = ConvertToURL("C:\Docs OpenOffice\LogoOpenOffice.png")
maPage.add(monImage)
resizeImageByWidth(monImage, 5700) ' largeur en 1/100 de mm
monImage.Position = positionImage
monImage.Name = "Logo2" ' donner un nom à cette image
End Sub
```

Retrouver, supprimer une image

L'image ayant été nommée, elle peut être retrouvée sur la page de dessin en utilisant la routine utilitaire de l'annexe B : `FindObjectByName`.

```
Rem Code14-02.sxc   bibli : lesImages Module3
Option Explicit

Sub SelectionnerImage()
Dim monDocument As Object, lesFeuilles As Object
Dim maFeuille As Object, maPage As Object
Dim monImage As Object
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("F01")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
monImage = FindObjectByName(maPage, "Logo1")
if IsNull(monImage) then
  print "Il n'existe aucune image de ce nom"
else
  monDocument.CurrentController.Select(monImage)
end if
End Sub
```

La méthode `remove` de l'objet page de dessin permet de supprimer une image :

```
maPage.remove(monImage)
```

Propriétés des images

Une image expose les propriétés des formes (voir le chapitre 13) et des propriétés spécifiques, dont les principales sont listées au tableau 14-1.

Tableau 14–1 Propriétés spécifiques des images

Propriété	Type	Signification
`AdjustRed`	`Integer`	Correction de la couleur Rouge, valeur en pourcentage positif ou négatif.
`AdjustGreen`	`Integer`	Correction de la couleur Vert.
`AdjustBlue`	`Integer`	Correction de la couleur Bleu.
`AdjustLuminance`	`Integer`	Correction de la luminosité, valeur en pourcentage positif ou négatif.
`AdjustContrast`	`Integer`	Correction du contraste, valeur en pourcentage positif ou négatif.
`Gamma`	`Double`	Valeur du Gamma.
`GraphicColorMode`	`Long`	Correspond au mode graphique sur la barre d'objets graphiques ; constante nommée.
`GraphicCrop`	`Object`	Recadrage de l'image ; structure.
`GraphicURL`	`String`	Chemin d'accès au fichier lié, ou référence du fichier intégré au document.

Les corrections ou le recadrage ne concernent que l'affichage dans le document ; elles ne modifient pas le fichier image.

La propriété `GraphicColorMode` reçoit une constante nommée (tableau 14-2) de la forme :

```
com.sun.star.drawing.ColorMode.NORMAL
```

Tableau 14–2 Constantes de GraphicColorMode

Constante	Mode graphique
`NORMAL`	Standard
`GREYS`	Niveaux de gris
`MONO`	Noir/Blanc
`WATERMARK`	Filigrane

La propriété GraphicCrop est une structure (du type com.sun.star.text.GraphicCrop) composée de quatre éléments de type Long :

- Top recadre le haut.
- Bottom recadre le bas.
- Left recadre le côté gauche.
- Right recadre le côté droit.

Il est nécessaire d'utiliser une variable intermédiaire pour modifier un élément.

```
dim rognure as object
rognure = monImage.GraphicCrop
rognure.Bottom = -2000
monImage.GraphicCrop = rognure
```

Une valeur positive rogne l'image, la partie visible s'étendant sur la surface offerte par les dimensions actuelles. Une valeur négative réduit l'image par rapport à ses dimensions, laissant une marge. Les valeurs sont exprimées en 1/100 de millimètres. Notez que l'interface utilisateur effectue simultanément un redimensionnement (boutons de choix Conserver l'échelle ou Conserver la taille de l'image).

Lorsqu'une image est intégrée au document, la propriété GraphicURL contient une URL particulière, par exemple :

```
vnd.sun.star.GraphicObject:10000000000001640000152463C5E09
```

Le nom du fichier image intégré dans le ZIP constituant le document est composé du nombre, suivi de l'extension du fichier. Le fichier est stocké dans le répertoire Pictures du ZIP.

Intégrer une image dans le document

Niklas Nebel, développeur à Sun Microsystems, a donné un moyen d'intégrer une image dans un document, au lieu d'insérer un lien. Il consiste à insérer une première fois l'image (n'importe où dans le document), puis à insérer l'image une deuxième fois en recopiant la propriété GraphicObjectFillBitmap obtenue de la première insertion, mais sans remplir la propriété GraphicURL. L'exemple est sur Draw, nous vous laissons déduire la méthode pour Writer et Calc.

```
Rem Code14-03.sxd   bibli : lesImages Module4
Option Explicit

Sub IntegrerImageFormatPNG()
Dim monDocument As Object, maPage As Object
Dim monImage As Object, ImageL As Object
Dim positionImage As New com.sun.star.awt.Point
```

```
monDocument = thisComponent
maPage = monDocument.DrawPages.getByName("Photos1")

ImageL = monDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
ImageL.GraphicURL = ConvertToURL("C:\Docs OpenOffice\Dolomites.jpg")
maPage.add(ImageL) ' cette image est liée

positionImage.x = 6500 ' 65 mm à droite du coin de la page
positionImage.y = 5300 ' 53 mm en dessous du coin de la page
monImage = monDocument.createInstance(_
          "com.sun.star.drawing.GraphicObjectShape")
monImage.GraphicObjectFillBitmap= ImageL.GraphicObjectFillBitmap
maPage.add(monImage) ' cette image est intégrée dans le document
maPage.remove(ImageL) ' supprimer l'image inutile
resizeImageByWidth(monImage, 11000) ' largeur en 1/100 de mm
monImage.Position = positionImage
End Sub
```

L'image `Dolomites.jpg` est disponible dans le même répertoire du zip téléchargeable. Elle fait 63 Ko. Le document initial grossira de 550 Ko, à cause de la conversion de l'image au format PNG. L'insertion sans lien est donc actuellement peu utile pour un document comportant de nombreuses images photographiques, car le fichier résultant serait de taille énorme.

Manipuler les équations

OpenOffice.org possède un éditeur permettant d'insérer des objets « équation » dans un document. Il s'agit d'`EmbeddedObjects` qu'il n'est pas possible de créer à l'aide de l'API, mais qui peuvent toutefois être manipulés. L'objet de notre exemple va être de modifier la taille de toutes les équations du document en cours.

```
Rem Code14-04.sxw   bibli : Equations Module1

Sub ModifierTailleEquations

dim NouvelleTaille As Integer
dim LesObjets As Object, LaFormule As Object, embObj As Object
dim i as long

'Demande la nouvelle taille
NouvelleTaille=InputBox("Donner une nouvelle taille :")

'Récupère la collection des objets insérés
LesObjets=ThisComponent.EmbeddedObjects
```

```
'boucle sur les objets du document
for i=0 to LesObjets.count-1
  LaFormule=LesObjets(i)
  'Est ce un objet inséré ?
  if LaFormule.supportsService("com.sun.star.text.TextEmbeddedObject") then
    embObj= LaFormule.embeddedObject
    'est ce une équation ?
    if embObj.supportsService("com.sun.star.formula.FormulaProperties") then
      embObj.BaseFontHeight= NouvelleTaille
      embObj.Modified= true
    endif
  endif
next i

End Sub
```

Cette macro commence par demander à l'utilisateur d'indiquer la nouvelle taille de la police des équations. Nous accédons ensuite à la collection des objets du document, que nous affectons à la variable `LesObjets`. En utilisant la propriété `count` de cette collection, nous bouclons sur chacun pour vérifier que l'objet inséré reconnaît bien le service d'objet texte inséré `TextEmbeddedObject`.

Pour être certain que nous manipulons bien un objet équation, nous accédons à l'objet lui-même par l'intermédiaire de la variable `embObj` et vérifions qu'il reconnaît le service `FormulaProperties`.

À ce stade, on est certain qu'il s'agit d'un objet équation. L'objet `embObj` expose les propriétés et méthodes qui lui sont propres. Ainsi, nous pouvons changer la taille de la police de base en affectant la propriété `BaseFontHeight`. En effet, la taille des différents symboles d'une équation mathématique est donnée en pourcentage par rapport à une taille de base. Ces paramètres de taille relative sont également accessibles et vous devriez pouvoir trouver leur nom sans difficulté en utilisant `xray` sur l'objet `embObj`. La propriété `Modified` permet d'informer le document que la représentation de la formule a changé et qu'il faut mettre à jour l'affichage.

Voici maintenant une macro permettant de changer l'équation elle même. Vous remarquerez que nous retrouvons l'équation à partir du nom de l'objet inséré. Ce nom est visible et modifiable avec le Navigateur.

Nous intervenons cette fois sur la propriété `Formula` qui attend un texte tel qu'il aurait été rentré dans l'éditeur d'équation.

Ces exemples ne sont en aucun cas exhaustifs. L'API couvrant les équations se situe dans le module `com.sun.star.formula` et le service `FormulaProperties`.

```
Rem Code14-04.sxw   bibli : Equations Module2

sub ModifierFormule

dim LesObjets As Object, LaFormule As Object

LesObjets= ThisComponent.EmbeddedObjects
LaFormule= LesObjets.getByName("Equation1")

LaFormule.embeddedobject.Formula= "{2+4} over {1+1} = 3"
LaFormule.embeddedObject.Modified= true
end sub
```

Conclusion

Ce chapitre clôt la présentation de l'API dans une optique de composants. Nous avons abordé dans ces chapitres la manipulation des quatre principaux types de document : Writer, Calc, Draw et Impress.

La quatrième partie de ce livre va élargir le recours aux macros à un contexte de développement applicatif à part entière. Les macros OpenOffice.org disposent pour cela d'un outil de création de boîte de dialogue performant, qui permet de bâtir de véritables environnements conviviaux pour l'utilisateur final. Les principes de base de ces boîtes de dialogue sont exposés dans le prochain chapitre.

Quatrième partie

Construire des applications avec OpenOffice.org

Dans cette partie, vous apprendrez à afficher des dialogues tout à fait semblables à ceux des applications classiques, à lire et écrire sur des bases de données, et à améliorer vos formulaires avec des macros. Vous y trouverez également quelques méthodes sophistiquées, sans prétendre bien sûr à l'exhaustivité.

15

Les boîtes de dialogue

Nous avons vu au chapitre 8 comment afficher des informations et obtenir une réponse de l'utilisateur. L'instruction `MsgBox` est parfaite pour afficher un court texte d'information ou une question fermée qui attende une réponse telle que Oui, Non ou Annuler. Quant à l'instruction `InputBox`, elle est d'un aspect peu esthétique, n'effectue aucun contrôle des données saisies et ne vous permet de demander qu'une information à la fois.

Grâce aux boîtes de dialogue, vous pourrez présenter de manière conviviale les différentes options de votre macro, recevoir plusieurs informations en un seul panneau, afficher des textes entiers, bref créer un véritable environnement applicatif pour votre macro.

> La documentation officielle sur les boîtes de dialogue se trouve dans le « Developer's Guide » au chapitre 11.5 du SDK. Ce chapitre comporte de nombreux liens hypertextes vers les descriptions IDL du SDK.

Construire une boîte de dialogue avec l'EDI

Ouvrez un nouveau document. Dans ce document et créez-y une nouvelle bibliothèque Basic que vous nommerez `PremierDialogue`. Dans l'EDI, cliquez droit sur la zone des onglets de modules et choisissez Insérer, Boîte de dialogue Basic. Vous obtenez la boîte de dialogue vierge de la figure 15-1.

Une autre méthode est décrite dans l'aide en ligne de Basic, section Guide : Création d'une boîte de dialogue Basic.

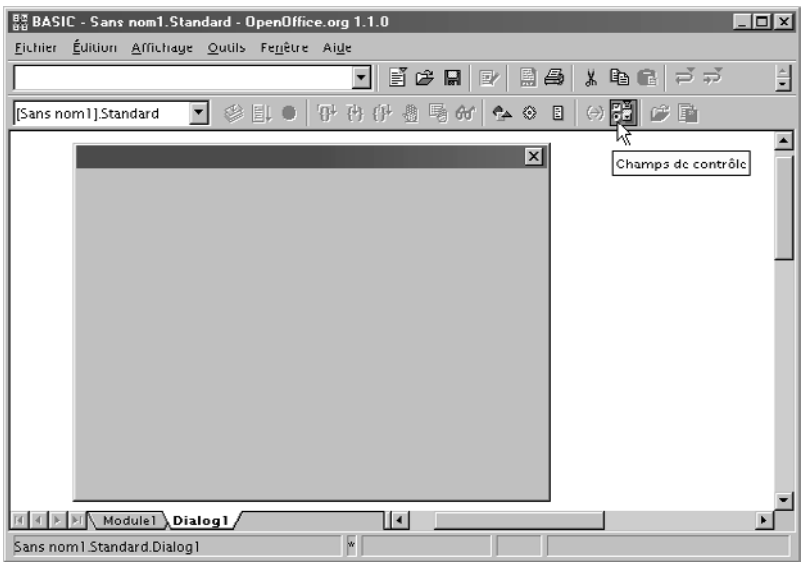


Figure 15–1
Une nouvelle boîte de dialogue

L'onglet Dialog1 peut être renommé comme pour un onglet de module. La fenêtre vierge au centre de la feuille est le panneau de la boîte de dialogue.

Dans la figure 15-1, le bouton pointé par le curseur de souris donne accès à une liste d'icônes ; un clic appuyé le transforme en barre flottante, représentée sur la figure 15-2. Chacune des icônes est un élément de dialogue, appelé Contrôle.

Le panneau de dialogue

Sélectionnez le panneau de dialogue avec la souris : pour cela, cliquez précisément sur un des bords du panneau. Des marques de redimensionnement apparaissent et le curseur de souris devient un curseur de déplacement ou un curseur de dimensionnement, voir la figure 15-2.

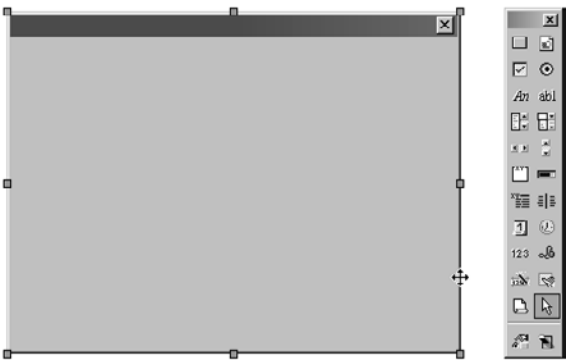


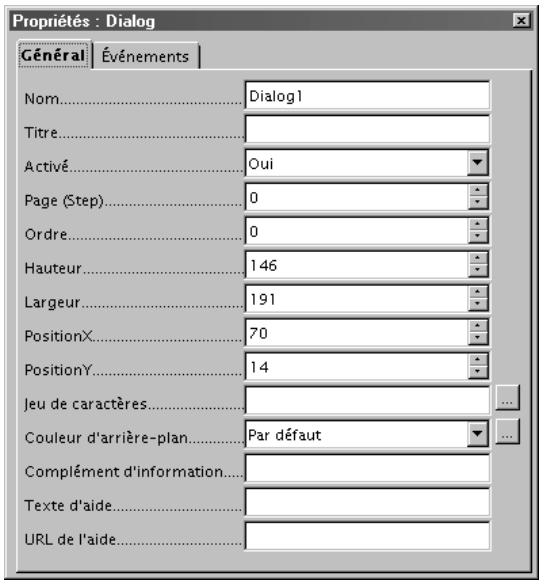
Figure 15–2
Sélection du panneau de dialogue et barre flottante des contrôles

La fenêtre des propriétés



Tout en gardant sélectionné le panneau de dialogue, faites un clic droit sur un bord et choisissez Propriétés. Vous obtenez une fenêtre flottante et redimensionnable, représentée sur la figure 15-3. Vous pouvez également y accéder depuis la barre flottante des contrôles, en cliquant sur l'icône tout en bas à gauche.

Figure 15–3
La fenêtre des propriétés



Vous aurez très souvent à utiliser la fenêtre des propriétés. Grâce à elle, vous allez inspecter et modifier les caractéristiques de l'objet sélectionné, dans le cas présent celles du panneau de dialogue, mais aussi celles de chacun des contrôles que vous allez y déposer. Comme cette fenêtre est encombrante n'hésitez pas à la redimensionner, la déplacer, la faire disparaître et réapparaître. Elle comporte deux onglets, Général et Événements.

L'onglet Général

La plupart des propriétés listées sur cet onglet existent aussi pour les autres contrôles. Les plus intéressantes pour le panneau dialogue sont :

- **Nom** - Le nom du dialogue, qui peut être différent du nom du module de dialogue ; inscrivez `DialogueUn`.
- **Titre** - Le texte qui apparaît dans la barre de titre du dialogue ; inscrivez ici `Mon premier dialogue`.
- **Hauteur, Largeur** - Les dimensions du panneau de dialogue, en *MapAppFont*.

- **PositionX, PositionY** - La position du coin haut/gauche de la boîte de dialogue, en *MapAppFont*. L'unité de mesure *Map AppFont* est liée aux hauteur et largeur moyennes des caractères de la police système. L'avantage est d'obtenir un panneau proportionné aux textes affichés par le système.

L'onglet Événements

En cliquant sur l'onglet Événements de la fenêtre des propriétés, vous verrez apparaître une liste (figure 15-4) :

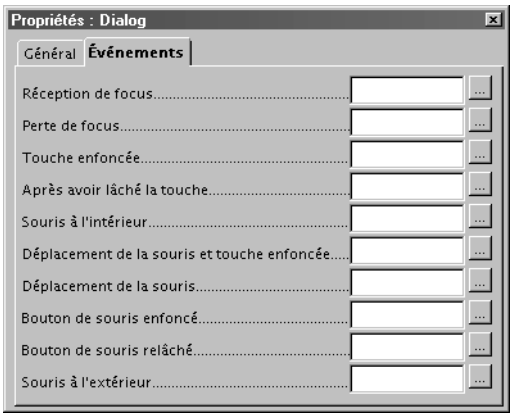


Figure 15–4
L'onglet Événements du panneau des propriétés

Cet onglet affiche la liste les différents événements concernant le panneau de dialogue. Vous obtiendrez une liste similaire pour chacun des contrôles que vous déposerez sur la boîte de dialogue.

L'onglet Événements sert à lancer une macro lorsqu'un événement particulier survient. Cette macro effectuera un petit traitement, par exemple mémoriser une information dans une variable pour un traitement ultérieur. La boîte de dialogue reste affichée, et selon les actions de l'utilisateur plusieurs événements seront peut-être déclenchés successivement, jusqu'à la fermeture de la boîte de dialogue.



Pour affecter une macro à un événement, cliquez sur la case à la droite de l'événement : le panneau de sélection s'affiche (figure 15-5).

1. Cliquez sur l'événement à traiter, la ligne se sélectionne.
2. Déroulez l'arbre des documents, puis des modules Basic dans le document.
3. Choisissez un module.
4. Choisissez une macro dans le module.
5. Cliquez sur le bouton Assigner.
6. Éventuellement, recommencez à l'étape 1 pour traiter un autre événement.
7. Cliquez sur le bouton OK.

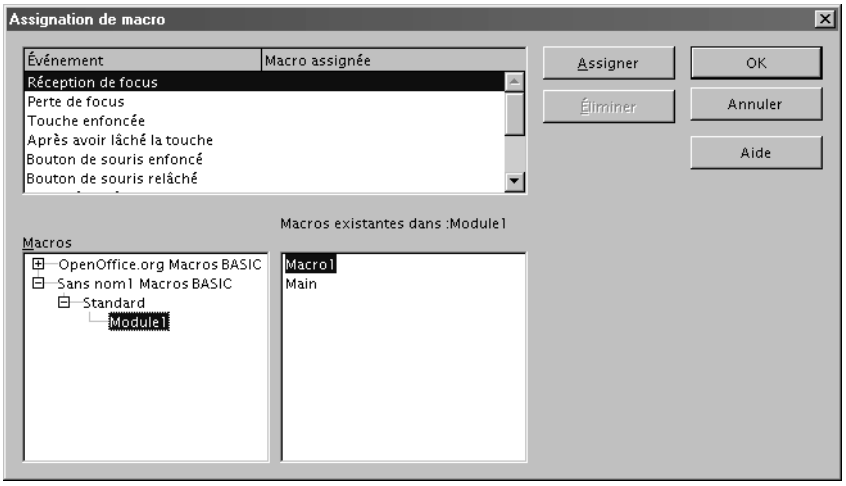


Figure 15–5
Affecter une macro à un événement

Si plus tard vous changez le nom de votre macro, pensez à réaffecter la macro à l'événement qu'elle doit traiter. Notez que la macro est appelée sans argument. Toutefois un objet `Event` est passé en argument pour peu que la déclaration de la macro le spécifie. Nous y reviendrons au chapitre 16.

Plus loin dans le présent chapitre, vous trouverez un exemple d'utilisation d'événement dans la description du contrôle Barre de Progression.

> **ATTENTION Les événements répétitifs**
>
> Certains événements sont déclenchés jusqu'à plusieurs fois par seconde, comme le déplacement de souris. Ce genre d'événement doit être traité seulement quand c'est absolument nécessaire, et avec des macros très courtes. Dans la plupart des cas, vous traiterez des événements plus simples, comme le déclenchement d'un bouton.

Votre première boîte de dialogue

Une boîte de dialogue se construit en déposant des contrôles un à un sur le panneau de dialogue. Nous commencerons par les contrôles les plus simples, qui ont aussi l'intérêt de présenter des propriétés que vous retrouverez dans la plupart des autres contrôles.

L'étiquette



Le contrôle Etiquette (Label) sert à afficher une information non modifiable par l'utilisateur. Dans l'EDI, cliquez sur l'icône Étiquette (An) dans la liste des contrôles, puis mettez la souris au-dessus du panneau de dialogue et cliquez-glissez pour obtenir un rectangle blanc allongé horizontalement. Il contiendra le texte Label1.

Faites apparaître la fenêtre des propriétés, en gardant la sélection du rectangle. Dans le champ de la propriété Titre, tapez le texte : « Voulez-vous lancer cette macro ? ». Tapez sur la touche Entrée ou cliquez sur le champ d'une autre propriété et vous devriez obtenir quelque chose de voisin de la figure 15-6.

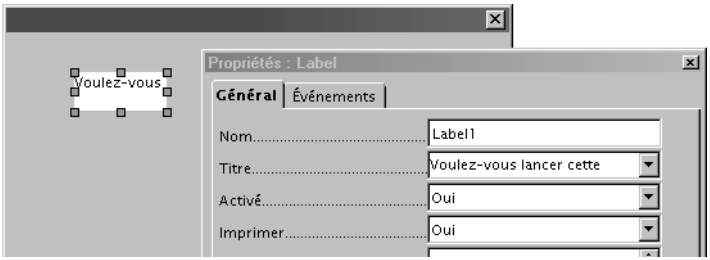


Figure 15–6
Insertion d'un contrôle étiquette

Vous avez certainement remarqué que le texte n'apparaît pas en totalité dans le champ Titre ; ce n'est pas grave, il suffit d'augmenter la largeur de la fenêtre des propriétés, ou de cliquer sur le triangle à droite du champ.

Cependant, le texte affiché dans le contrôle est tronqué, lui aussi ! Avec la souris, il suffit d'augmenter la largeur du rectangle. Toujours avec la souris, exercez-vous à redimensionner et déplacer ce contrôle, auquel Basic a donné le nom Label1, mais que vous pouvez renommer à votre convenance. Vous pouvez également modifier directement les propriétés Hauteur, Largeur, PositionX, PositionY que l'on retrouve aussi sur tous les contrôles.

Le contrôle Label sert uniquement à afficher un texte fixe. Néanmoins, il a plusieurs apparences que vous allez découvrir en jouant avec les propriétés.

- **Alignement** - Centrez le texte dans son rectangle.
- **Cadre** - Essayez les différentes possibilités.
- **Couleur d'arrière-plan** - Essayez des couleurs claires.
- **Jeu de caractères** - Changez la police, ou sa taille.
- **Plusieurs lignes** - Choisissez Oui, puis redimensionnez le rectangle en un carré plus petit. Vous pouvez forcer un retour à la ligne dans le texte, avec la touche Maj-Entrée.

En fonction de vos besoins, vous décidez donc de l'aspect le plus approprié de votre contrôle. Ceci ne change absolument rien dans le codage de votre macro. La preuve : nous n'avons encore rien programmé !

Le Bouton



Un contrôle Bouton (Button) sert à déclencher un traitement lorsque l'utilisateur l'actionne. De la même manière que précédemment, choisissez l'icône Bouton dans la liste des contrôles et dessinez un rectangle sur le panneau du dialogue. Affichez les propriétés.

Le bouton obtenu est nommé par Basic : CommandButton1.

Le bouton Standard

La propriété Type de Bouton est initialement à Standard. Un bouton standard sert à déclencher un événement, qui doit alors être pris en compte avec une routine spécifique. Nous indiquerons comment faire dans le chapitre 16. Pour le moment, nous nous contenterons de l'usage le plus fréquent et le plus simple.

Le bouton OK

Changez le titre du bouton en Oui. Changez le type de bouton en OK. Vous venez de créer un bouton spécial : quand l'utilisateur clique sur ce bouton, le dialogue se termine avec une valeur de retour spécifique.

Dans le champ Texte d'aide des propriétés du bouton Oui, tapez : « Exécuter le dialogue ». Nous verrons plus loin que ce texte apparaîtra sous forme de bulle au passage de la souris.

Le bouton Annuler

Glissez encore une fois la souris sur le panneau de dialogue pour obtenir un deuxième bouton. Le bouton obtenu est nommé par Basic : CommandButton2. Changez le titre du bouton en Non. Changez le type de bouton en Annuler. Vous venez de créer un autre bouton spécial : quand l'utilisateur clique sur ce bouton, le dialogue se termine avec une autre valeur de retour spécifique. En fait, ce bouton effectue exactement la même action que de cliquer sur la case de fermeture de fenêtre.

Le bouton Aide

Il existe un dernier type de bouton, le bouton Aide. Dans son fonctionnement de base, il permet d'afficher l'aide en ligne d'OpenOffice.org.

Ajuster les éléments du dialogue

Exercez-vous à sélectionner un des contrôles de la boîte : il suffit de cliquer dessus. Déplacez-le en glissant avec la souris. Redimensionnez-le. Pour sélectionner plusieurs contrôles à la fois essayez deux méthodes :

- un clic sur le premier contrôle, un Maj-clic sur les autres ;



- dans la barre flottante des contrôles, choisissez l'icône Sélection ; cliquez en dehors d'un contrôle et faites glisser la souris pour créer un rectangle pointillé englobant les contrôles à sélectionner.

Quand un groupe de contrôles est sélectionné, vous pouvez le déplacer sans modifier leurs positions relatives, ou effectuer un redimensionnement global.

La fenêtre des propriétés affiche les valeurs de propriétés qui sont identiques pour tous les objets du groupe. Cela peut servir à modifier globalement la taille des boutons, par exemple.

Attention, si vous sélectionnez uniquement le panneau de dialogue, son déplacement laisse les contrôles sur place ! Donc sélectionnez l'ensemble avec la technique du rectangle pointillé, puis déplacez le groupe.

Tester le dialogue

Après quelques ajustements, vous obtiendrez un dialogue similaire à celui de la figure 15-7.



Figure 15–7
Votre première boîte de dialogue



C'est le moment de tester l'aspect final de votre dialogue. Dans la liste des contrôles, cliquez sur l'icône tout en bas à droite.

Le panneau de dialogue s'affiche. Déplacez la souris au-dessus du bouton Oui, une bulle d'aide apparaît. Cliquez sur un des boutons, ou sur la croix de fermeture de fenêtre et le dialogue disparaît.

L'intérêt du test est de voir votre dialogue tel qu'il se présentera à l'utilisateur. Et ceci sans avoir encore écrit une seule ligne de codage !

Évidemment, il n'y a aucune exécution réelle. Aucun événement n'est produit dans ce mode de test de l'interface.

Utilisez l'EDI pour changer sur le bouton Non la propriété Type de Bouton en : Standard. Testez de nouveau : le panneau de dialogue reste présent quand vous cliquez sur ce bouton. Un bouton de type standard ne fait rien par lui-même, il faut affecter une macro à l'événement Lors du déclenchement ; nous en verrons un exemple plus loin.

Cliquez sur l'autre bouton ou sur la case de fermeture de fenêtre. Dans l'EDI, remettez le type de bouton sur Annuler.

Exécuter le dialogue

Dans la même bibliothèque PremierDialogue, écrivez dans un module le code suivant :

```
rem Code15-01.sxw  bibli : PremierDialogue
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
' la bibliothèque est déjà en mémoire
' car cette macro est dans la même bibliothèque
' récupérer l'objet bibliothèque
bibli = DialogLibraries.GetByName("PremierDialogue")
' récupérer la boîte de dialogue dans la bibliothèque
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue) ' créer le dialogue
if Dlg.Execute = exitOK then
  MsgBox("Macro Main1 de la bibli PremierDialogue")
end if
Dlg.Dispose
End Sub
```

La première étape consiste à obtenir l'objet bibliothèque appelé PremierDialogue. Tous les objets bibliothèques du document sont connus de l'objet DialogLibraries.

Cette bibliothèque peut comporter plusieurs boîtes de dialogue. L'instruction suivante obtient l'objet boîte de dialogue Dialog1. Notez qu'il s'agit du nom de l'onglet dans l'EDI, et non pas de la propriété Nom du panneau de dialogue.

> Veillez à bien respecter la casse pour le nom de la bibliothèque et le nom du dialogue.

L'instruction CreateUnoDialog sert à obtenir un objet capable d'afficher la boîte de dialogue. La fonction Dlg.Execute affiche la boîte et gère toutes les interactions de l'utilisateur jusqu'à la fermeture de la boîte de dialogue. Cette fonction retourne une valeur numérique. Ici, nous testons si le dialogue a été fermé par un bouton de type OK. Si le dialogue a été fermé par un bouton de type Annuler, la valeur de retour est :

```
com.sun.star.ui.dialogs.ExecutableDialogResults.CANCEL
```

MÉTHODE

Pour information, la valeur de retour pour OK est 1, mais ceci est un choix d'implémentation. Utilisez le plus souvent possible la constante API et non la valeur numérique. Cette habitude amènera une plus grande facilité de relecture.

La dernière instruction libère les ressources de dialogue. Elle n'est pas nécessaire ici car la fin d'exécution va libérer toutes les ressources, mais c'est une bonne habitude de codage.

Accéder aux bibliothèques de dialogues

L'accès à une boîte de dialogue située dans une autre bibliothèque que le sous-programme qui l'utilise, nécessite de charger auparavant cette bibliothèque. Ceci est très similaire à ce que nous avons rencontré pour les appels de routines de bibliothèques au chapitre 7.

Si votre code se trouve dans la bibliothèque standard de votre document et le dialogue dans la bibliothèque `PremierDialogue`, vous devez d'abord charger cette bibliothèque, ce qui donne ceci :

```
rem Code15-01.sxw  bibli : Standard
Option Explicit

Sub Main2()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
' charger la bibliothèque en mémoire
DialogLibraries.LoadLibrary( "PremierDialogue" )
' récupérer l'objet bibliothèque
bibli = DialogLibraries.GetByName("PremierDialogue")
' récupérer la boîte de dialogue dans la bibliothèque
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue) ' créer le dialogue
if Dlg.Execute = exitOK then
  MsgBox("Macro Main2 de la bibli Standard")
end if
Dlg.Dispose
End Sub
```

Enfin, si votre dialogue se trouvait dans une bibliothèque de `soffice` autre que `Standard`, le code de chargement de bibliothèque deviendrait :

```
' charger la bibliothèque en mémoire
GlobalScope.DialogLibraries.LoadLibrary("PremierDialogue")
' récupérer l'objet bibliothèque
bibli = GlobalScope.DialogLibraries._
        GetByName("PremierDialogue")
```

Précision : dans l'exemple PremierDialogue, il n'est pas nécessaire de charger la bibliothèque dialogue parce que la macro est lancée depuis l'EDI. Dans ce cas, la bibliothèque des macros et la bibliothèque soeur des dialogues ont été chargées par OOo à l'ouverture de l'EDI. Il en est de même si on lance la macro directement par le menu Outils>Macros, qui charge aussi la bibliothèque.

En revanche, si on ouvre le document sans ouvrir l'EDI, les bibliothèques ne sont pas chargées. Si la macro est lancée par un bouton, OOo charge bien la bibliothèque de macros, mais pas celle des dialogues. Il faut alors la charger par programmation (méthode `LoadLibrary`), même si cette boîte de dialogue est dans `Standard`.

Les principaux champs de saisie

Jusqu'à présent, nous n'avons qu'un dialogue équivalent à un `MsgBox`. Néanmoins, en ajoutant de nouveaux contrôles, nous allons multiplier les possibilités d'interaction avec l'utilisateur.

La zone de texte

Le contrôle Zone de texte (`TextField`) permet à l'utilisateur de saisir un texte. Dans un nouveau document, créez une nouvelle boîte de dialogue avec :

- un bouton OK,
- un bouton Annuler,
- un contrôle Étiquette intitulé : `Votre Prénom`,
- juste en-dessous, un contrôle Zone de texte ; c'est l'icône Zône de texte (abl) située à droite du contrôle Étiquette sur la barre flottante des contrôles. Dans la propriété Texte de ce contrôle inscrivez : `Arthur`.

Vous obtenez une boîte de dialogue similaire à celle de la figure 15-8.



Figure 15–8
Dialogue avec une zone de texte

Vous retrouvez les propriétés d'un contrôle Étiquette et de nouvelles propriétés. Utilisez l'icône de test pour voir leur utilité. Dans l'état initial, le test vous montrera que vous pouvez modifier le champ zone de texte.

Donnez à la propriété Longueur de texte max la valeur `5`. Le champ n'affiche plus que `Arthu` et vous êtes limité à 5 caractères. Remettez la valeur zéro pour ne pas avoir de limitation. Donnez à la propriété En lecture seule la valeur `Oui`. Durant le test, vous ne pouvez plus modifier le champ. Remettez la valeur `Non`. Donnez à la propriété Caractère pour mots

de passe la valeur x. Le contenu du champ devient xxxxxx et tout ce que vous tapez en mode test est affiché avec le caractère x. Effacez ce caractère dans la propriété, le contenu réapparaît en clair.

Les propriétés Plusieurs lignes, Barre de défilement horizontale et Barre de défilement verticale vous seront utiles avec un champ zone de texte contenant de nombreuses lignes. En effet, ce champ affiche le contenu d'une variable `String` qui peut atteindre 65 535 caractères. Vous pouvez introduire des retours à la ligne avec la touche Maj-Entrée.

Une zone de texte en lecture seule avec la propriété Plusieurs lignes est pratique pour afficher un texte d'information tel qu'un mode d'emploi. Avantages par rapport au contrôle Étiquette : les barres de défilement permettent de lire un texte très long ; l'utilisateur peut sélectionner et copier tout ou partie de ce texte.

Comment récupérer le texte saisi par l'utilisateur du dialogue ? Quelques instructions nouvelles vont suffire.

```
rem Code15-02.sxw  bibli : ZTexte Module1
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champPrenom As Object, prenom As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Ztexte")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champPrenom = Dlg.GetControl("TextField1")
  prenom = champPrenom.Text
  MsgBox("Votre prénom est : " & prenom)
end if
Dlg.Dispose
End Sub
```

Nous récupérons d'abord dans la variable `champPrenom` l'objet contrôle zone de texte, grâce à la fonction `GetControl` de l'objet `Dlg`. La fonction `getControl` d'un objet dialogue est très utilisée car elle permet d'accéder à chacun des contrôles de la boîte de dialogue. Le nom du contrôle passé en argument doit être écrit en respectant la casse. Le texte frappé par l'utilisateur (ou le texte initial, s'il n'a pas été modifié) est obtenu avec la propriété `Text` sous la forme d'une variable `String`. Si le contrôle utilise la propriété Caractère pour mots de passe, le texte est récupéré en clair.

Le champ numérique

Le contrôle Champ numérique (`NumericField`) sert à récupérer une valeur numérique saisie par l'utilisateur. Dans un nouveau document, créez une nouvelle boîte de dialogue avec :

- un bouton OK, un bouton Annuler,
- un contrôle Étiquette intitulé : `Votre Poids`,
- juste en-dessous, un contrôle Champ numérique ; c'est l'icône 123 sur la barre flottante des contrôles.

Le contrôle Champ numérique possède plusieurs propriétés spécifiques. Remplissez-en quelques-unes :

- Valeur = 70 ;
- Valeur min = 3 ;
- Valeur max = 250 ;
- Décimales = 1.

Vous avez remarqué que la fenêtre des propriétés ajoute un nombre de décimales zéro correspondant à la valeur courante de la propriété Décimales. Le caractère séparateur de décimales est bien une virgule, conformément à l'usage français. Vous obtenez une boîte de dialogue similaire à celle de la figure 15-9.

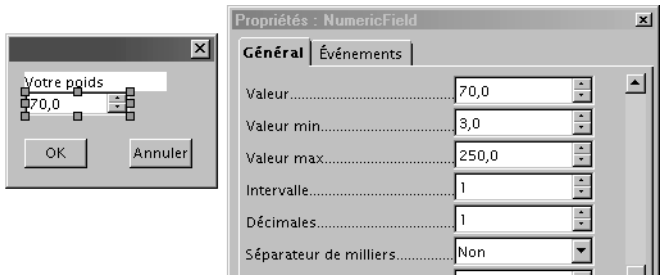


Figure 15–9
Dialogue avec un Champ numérique

Jouez avec les propriétés. Utilisez l'icône de test pour voir leur utilité. Modifiez à `Oui` la propriété Compteur : les triangles qui apparaissent servent à incrémenter ou décrémenter la valeur numérique. La valeur numérique fournie par l'utilisateur est récupérée de manière très similaire à un contrôle zone de texte.

```
rem Code15-03.sxw  bibli : Znum
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champPoids As Object, lePoids As Double

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
```

```
bibli = DialogLibraries.GetByName("Znum")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champPoids = Dlg.GetControl("NumericField1")
  lePoids = champPoids.Value
  MsgBox("Votre poids est : " & lePoids & " Kg")
end if
Dlg.Dispose
End Sub
```

La valeur obtenue est de type `Double`, même si vous avez seulement besoin d'un nombre entier. Il existe d'autres contrôles numériques spécialisés (heure, date, monétaire, masqué) que nous étudierons plus loin dans ce chapitre.

La zone de liste

Le contrôle Zone de liste (`ListBox`) permet à l'utilisateur de choisir un texte parmi plusieurs proposés. On dit aussi : choisir une entrée parmi plusieurs. Il existe plusieurs variations de zone de liste, qui ont beaucoup de points communs.

La zone de liste simple

Dans cette variante, l'utilisateur ne peut pas modifier les textes. Dans un nouveau document, créez une nouvelle boîte de dialogue avec :

- un bouton OK, un bouton Annuler,
- un contrôle Étiquette intitulé : `Votre destination`,
- juste en-dessous, un contrôle Zone de liste.



Le contrôle zone de liste possède plusieurs propriétés spécifiques. Remplissez-en quelques-unes :

- Entrées de liste = tapez ici 7 ou 8 noms de ville, avec un retour à la ligne entre chacune (touche Maj-Entrée) ;
- Déroulante = Oui ou Non ;
- Nombre de lignes = 5 ;
- Sélection multiple = Non.

Selon la valeur de la propriété Déroulante, vous obtenez une des deux boîtes de dialogue de la figure 15-10.

Utilisez l'icône Test pour voir le comportement dynamique du dialogue. Si vous mettez Non à la propriété Déroulante et augmentez suffisamment la hauteur du contrôle, toutes les lignes disponibles sont affichées, ce qui est parfois plus pratique pour l'utilisateur. La propriété Nombre de lignes n'est utile que pour une liste déroulante ; elle précise le nombre de lignes visibles dans la section déroulée.

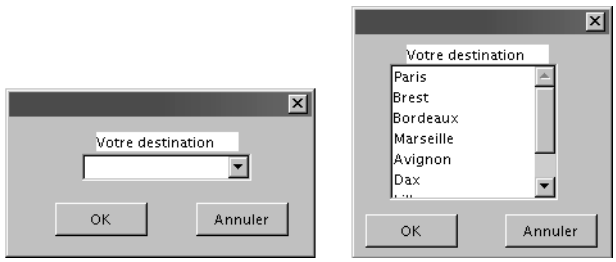


Figure 15–10
Dialogue avec une zone de liste simple

Tapez une lettre. S'il existe une entrée commençant par cette lettre, elle s'affiche ; s'il en existe plusieurs, la première s'affiche. C'est une recherche incrémentale : si vous tapez plusieurs lettres successivement, la première entrée commençant par ces lettres s'affiche. Appuyez sur la barre Espace pour reprendre la recherche à zéro.

La récupération du choix de l'utilisateur nécessite plusieurs paramètres.

```
rem Code15-04.sxw  bibli : Zliste Module1
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champDest As Object
Dim dest As String, numdest As Integer

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Zliste")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
champDest = Dlg.GetControl("ListBox1")
champDest.SelectItemPos(1, True)
if Dlg.Execute = exitOK then
  dest = champDest.SelectedItem
  numdest = champDest.SelectedItemPos
  MsgBox("Rang choisi : " & numdest & " = " & dest)
end if
Dlg.Dispose
End Sub
```

La méthode `SelectItemPos` initialise l'affichage en sélectionnant un des choix. Chaque choix a une position, le premier choix ayant la position zéro. Vous devez effectuer l'initialisation avant d'afficher le dialogue, mais après l'avoir créé. Le premier paramètre de la méthode est la position, le deuxième précise si ce choix est sélectionné ou non.

La propriété `SelectedItem` est une chaîne de caractères contenant le choix de l'utilisateur. La propriété `SelectedItemPos` est la position de ce choix dans la liste présentée. Il peut être plus simple d'utiliser la position que la chaîne de caractères.

La zone de liste à sélection multiple

Si dans le contrôle zone de liste la propriété Sélection multiple vaut `Oui`, l'utilisateur peut sélectionner plusieurs lignes (en utilisant la souris comme dans l'explorateur de fichiers). La propriété Déroulante est positionnée à `Non`.

ROBUSTESSE

Attention, l'utilisateur peut tout désélectionner, en faisant un Ctrl + clic sur la dernière ligne encore sélectionnée. Il faut en tenir compte dans le codage.

Le codage est différent car on obtient un tableau des valeurs sélectionnées au lieu d'une seule valeur. Les propriétés ont un nom quasiment identique : `Item` (élément) devient `Items` pour marquer le pluriel.

```
rem Code15-04.sxw  bibli : Zliste Module3
Option Explicit

Sub Main3()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champDest As Object
Dim dest() As String, numdest() As Integer, x As Integer
Dim listeChoix As String, nbSelectDest As Integer

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Zliste")
monDialogue = bibli.GetByName("Dialog3")
Dlg = CreateUnoDialog(monDialogue) ' créer le dialogue
champDest = Dlg.GetControl("ListBox1")
champDest.SelectItemPos(1, True)
champDest.SelectItemPos(4, True)
if Dlg.Execute = exitOK then
  dest = champDest.SelectedItems ' attention au pluriel !
  numdest = champDest.SelectedItemsPos ' pluriel !
  nbSelectDest = UBound(numdest()) +1
  MsgBox("Nombre de destinations choisies : " & nbSelectDest)
  listeChoix = "Votre choix :"
  for x = 0 to UBound(numdest())
    listeChoix = listeChoix & "  " & dest(x)
  next
  if nbSelectDest > 0 then  MsgBox(listeChoix)
end if
Dlg.Dispose
End Sub
```

Nous avons choisi de sélectionner au départ deux lignes. On récupère un tableau de `String` avec `SelectedItems`, et un tableau d'`Integer` avec `SelectedItemsPos`. La valeur

minimale de l'index des tableaux est toujours zéro, `UBound` fournit la valeur haute de l'index. Notez que `UBound` renvoie -1 si rien n'est sélectionné.

La zone de liste combinée



Dans cette variante, l'utilisateur peut fournir un texte autre que ceux proposés. Créez une boîte de dialogue identique à celle de la zone de liste simple, mais choisissez l'icône zone combinée.

L'apparence du dialogue est identique à celle de la zone de liste simple (figure 15-10).

Inscrivez dans la propriété `Texte` le nom d'une des destinations de la liste, par exemple `Brest`. Activez le test du dialogue et déroulez la liste : l'entrée `Brest` est sélectionnée. Il en est de même si vous saisissez le nom d'une des autres destinations ; et en la saisissant, vous verrez la recherche incrémentale fonctionner.

Les propriétés `SelectedItemPos`, `SelectedItemsPos`, `SelectedItem`, `SelectedItems` n'existent pas pour une zone de liste combinée. La méthode `SelectItemPos` non plus. La récupération du choix de l'utilisateur se fait donc de manière différente de celle vue pour la zone de liste simple.

```
rem Code15-04.sxw  bibli : Zcombi Module1
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champDest As Object
Dim dest As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Zcombi")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
champDest = Dlg.GetControl("ComboBox1")
' champDest.Text = "Marseille"
if Dlg.Execute = exitOK then
  dest = champDest.Text
  MsgBox("Destination choisie : " &  dest)
end if
Dlg.Dispose
End Sub
```

Que l'utilisateur choisisse une des propositions ou en indique une autre, on récupère le texte de l'entrée dans la propriété `Text`, de type `String`. La ligne de code en commentaire juste avant `Dlg.Execute` montre comment on peut, à l'exécution, choisir la valeur par défaut.

Il appartient au programmeur d'ajouter éventuellement à la liste une nouvelle entrée fournie par l'utilisateur. Ceci sera développé dans le chapitre 16.

Les cases à cocher

Une Case à cocher (CheckBox) sert à rentrer une information binaire (oui, non). L'utilisateur change l'état de la case en cliquant dessus. Créez une nouvelle boîte de dialogue avec :



- un bouton OK, un bouton Annuler,
- trois contrôles Case à cocher ayant respectivement pour titre `Gras`, `Italique`, `Souligné`.

En sélectionnant les trois contrôles cases à cocher dans l'EDI, donnez-leur la même largeur, la même hauteur et la même PositionX. En utilisant la PositionY de chacun, espacez-les régulièrement de haut en bas. Pour la case `Italique`, mettez la propriété Statut à : `Sélectionné`. Vous obtenez la boîte de dialogue de la figure 15-11.



Figure 15–11
Dialogue avec des Cases à cocher

Testez la boîte de dialogue : chaque case peut être cochée ou décochée par l'utilisateur. Le codage ci-dessous vous montre une manière, parmi d'autres, de savoir quelles cases ont été cochées.

```
rem Code15-05.sxw   bibli : Cocher Module1
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim coche As Object
Dim Gras As Boolean, Italique As Boolean, Souligne As Boolean

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Cocher")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  coche = Dlg.GetControl("cocheGras")
  Gras = (coche.State = 1)
  coche = Dlg.GetControl("cocheItalique")
  Italique = (coche.State = 1)
  coche = Dlg.GetControl("cocheSouligne")
  Souligne = (coche.State = 1)
```

```
  Print "Texte";
  if Gras then Print " gras";
  if Italique then Print " italique";
  if Souligne then Print " souligné";
  if not (Gras or Italique or Souligne) then Print " normal";
  Print
end if
Dlg.Dispose
End Sub
```

La variable `coche` donne accès successivement à chaque contrôle case à cocher ; la propriété State (État) du contrôle peut prendre une des valeurs :

0 la case n'est pas cochée ;

1 la case est cochée ;

2 état « indéterminé » ou « je ne sais pas ».

L'étrange état 2 ne peut être atteint que si la propriété Statut triple a la valeur `Oui`. Dans ce cas, la coche apparaît en grisé. Dans la grande majorité des utilisations, Statut triple est laissé à `Non`, et State ne peut alors valoir que 0 ou 1. Notre codage remplit pour chaque contrôle une variable `Boolean` à partir de State. Ces variables servent ensuite au traitement.

Les cases de choix 1 parmi N

Ces cases de choix (`OptionButton`) sont parfois appelées « Bouton radio » par référence aux vieux postes de radio sur lesquels, quand on enfonçait un bouton pour changer de gamme d'ondes, un autre bouton précédemment enfoncé remontait par un effet mécanique. L'intérêt des Cases de choix 1 parmi N est que le système vous assure qu'une seule case sera active dans l'ensemble des cases de choix.

Créez une nouvelle boîte de dialogue avec :

- un bouton OK, un bouton Annuler,
- quatre contrôles Case de choix 1 parmi N ayant respectivement pour titre : `Célibataire`, `Marié`, `Veuf`, `Divorcé`.

En sélectionnant les quatre contrôles Cases de choix dans l'EDI, donnez-leur la même largeur, la même hauteur et la même PositionX. En utilisant la PositionY de chacun, espacez-les régulièrement de haut en bas.

Sur une des quatre cases, modifiez la propriété Statut à : `Sélectionné`, afin d'assurer qu'une des possibilités est choisie au départ.

Utilisez la fonction test pour changer à volonté la case choisie : il est effectivement impossible de sélectionner plus d'une case. La figure 15-12 vous montre le résultat.



Figure 15–12
Dialogue avec des Cases de choix 1 parmi N

Dans une boîte de dialogue plus complète, vous pourriez avoir cette série de cases et une autre série, par exemple pour le choix Homme, Femme. Comment Basic va-t-il distinguer les deux ensembles de cases ?

La réponse est la suivante : Basic regroupe dans un ensemble toutes les cases 1 parmi N qui possèdent des valeurs successives de leur propriété Ordre. La valeur de cette propriété est incrémentée par l'EDI chaque fois qu'on dépose un nouveau contrôle. Si vous souhaitez mettre un deuxième ensemble de cases, déposez sur la boîte un autre type de contrôle (éventuellement invisible), puis déposez successivement les cases du deuxième ensemble.

Il est possible de modifier la valeur de la propriété Ordre d'un contrôle afin de le remettre dans une série ; cependant comme l'EDI renumérote alors une partie des contrôles, il faudra plusieurs essais pour obtenir un ordre correct. À chaque essai, utilisez le test pour juger du résultat. En conclusion, prévoyez à l'avance vos contrôles 1 parmi N ; déposez-les successivement, déposez un contrôle visuel (décrit plus loin), puis déposez les contrôles de la série suivante.

Voici un exemple de code qui traite le dialogue de la figure 15-12.

```
rem Code15-06.sxw   bibli : Choix1N
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim coche As Object
Dim Celib As Boolean, Marie As Boolean
Dim Divorce As Boolean, Veuf As Boolean

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Choix1N")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  coche = Dlg.GetControl("_celib")
  Celib = coche.State
  coche = Dlg.GetControl("_mar")
  Marie = coche.State
  coche = Dlg.GetControl("_veuf")
  Veuf = coche.State
```

```
  coche = Dlg.GetControl("_div")
  Divorce = coche.State
  if Divorce then Print "Vous êtes divorcé"
  if Marie then Print "Vous êtes marié"
  if Veuf then Print "Vous êtes veuf"
  if Celib then Print "Vous êtes célibataire"
end if
Dlg.Dispose
End Sub
```

Ici, la propriété `State` des contrôles de choix 1 parmi N nous donne directement une valeur binaire, `True` quand le contrôle est sélectionné (et donc que tous les autres du groupe sont `False`). À l'exécution de ce codage, une instruction `Print` et une seule sera exécutée si l'utilisateur a fermé la boîte de dialogue par le bouton `OK`.

REMARQUE **Zone de liste, choix 1 parmi N**

La zone de liste et un groupe de cases de choix 1 parmi N remplissent la même fonction de deux manières différentes, visuellement et dans le codage. Le choix de l'une ou l'autre des méthodes dépend du contexte et des contraintes d'interface utilisateur.

L'aspect visuel des boîtes de dialogue

Cela peut vous sembler futile, mais l'aspect visuel et l'ergonomie d'un dialogue jouent un grand rôle dans l'impression ressentie par l'utilisateur.

Présenter un dialogue ergonomique signifie qu'il doit paraître clair, avec des possibilités et des comportements similaires à ceux que l'utilisateur peut avoir déjà expérimentés ailleurs. Regardez avec un œil de concepteur de dialogue les divers panneaux affichés par les applications actuelles sous interface graphique. Observez en particulier les différents panneaux des options d'une application, qui vous donneront des exemples de dialogues complexes et pensés pour être ergonomiques.

Les sections suivantes de ce chapitre vont décrire des éléments à ne pas négliger.

Cohérence des dimensions

Les utilisateurs ont parfois des écrans plus réduits que le vôtre. En règle générale et avec l'état actuel du parc informatique, vos boîtes de dialogue doivent pouvoir tenir facilement dans un écran SVGA (800 × 600 pixels). Si vous avez la chance d'utiliser un écran 19 pouces en 1280 × 1024, passez-le de temps en temps dans la définition 800 × 600 et vérifiez que la boîte de dialogue ne déborde pas de l'écran.

Quand vous avez des éléments répétitifs, comme plusieurs boutons, ou plusieurs cases à cocher, veillez à les espacer régulièrement et leur donner les mêmes dimensions. Si votre application emploie plusieurs boîtes de dialogue, prenez des dimensions identiques pour les contrôles de même type, dans la mesure du possible.

La notion de « Focus »

Difficile de traduire le terme anglais *Focus* ! Dans le contexte des systèmes d'exploitation à fenêtres, dont MS-Windows est le plus connu, un élément visuel a le *Focus* quand il est en activité, au premier plan. Pour une fenêtre, ceci est manifesté par une couleur particulière de l'en-tête. Cependant, dans une fenêtre, chaque contrôle peut, tour à tour, prendre le *Focus*. C'est alors ce contrôle qui reçoit les saisies de l'utilisateur.

Dans une fenêtre de dialogue, le *Focus* passe d'un contrôle à l'autre en cliquant sur un contrôle ou en appuyant sur la touche Tab ou Maj - Tab, ou parfois en appuyant sur une autre touche. Un contrôle en *Focus* se distingue par certaines modifications visibles, par exemple un champ texte devient sélectionné dans une couleur soutenue, ou le libellé d'un bouton est encadré en pointillé. Quand un bouton a le *Focus*, appuyer sur la touche Entrée déclenche celui-ci.

L'ordre dans lequel les contrôles sont parcourus avec des tabulations successives est celui des valeurs croissantes de la propriété Ordre. Les valeurs successives sont choisies par l'EDI dans l'ordre de dépôt des contrôles sur la boîte de dialogue. En imposant des valeurs adéquates, vous pouvez choisir un ordre de parcours quelconque. Cependant, comme l'EDI modifie parfois l'ordre d'autres contrôles pour garder une cohérence, plusieurs modifications seront peut-être nécessaires. Enfin n'oubliez pas que la propriété Ordre est utilisée pour regrouper les contrôles de Choix 1 parmi N.

La propriété TabStop indique si le contrôle peut ou non recevoir le *Focus*. En général, un contrôle en lecture seule ne reçoit pas le *Focus*.

La fonction de test d'un dialogue vous permet de vérifier, par tabulations successives, l'ordre de passage du *Focus* sur vos contrôles.

Lettre accélératrice

Sur chaque bouton d'une boîte de dialogue, une lettre de son libellé est soulignée. Il suffit de taper celle-ci pour amener le focus sur ce bouton, à condition que le focus soit déjà sur un autre bouton. Par défaut, OpenOffice.org choisit la lettre accélératrice en fonction des libellés des boutons. À la conception de la boîte de dialogue, vous pouvez imposer la lettre accélératrice d'un bouton en la faisont précéder par le caractère tilde ~.

Le bouton par défaut

Dans les propriétés des boutons, il en existe une appelée Bouton par défaut. Dans un dialogue comportant plusieurs boutons, si vous mettez à Oui cette propriété pour un (et un seul) des boutons, celui-ci sera déclenché si vous appuyez la touche Entrée.

Néanmoins, ceci se fera *à condition que le focus ne soit sur aucun des boutons* ; typiquement, le *focus* est sur un champ de saisie que vous venez de remplir. Si le *focus* est sur un des boutons, il sera déclenché avec la touche Entrée, même s'il n'est pas le bouton par défaut. Cette condition illustre aussi l'importance de l'ordre des contrôles, qui gouverne la succession des *focus*.

Dans la figure 15-11 (voir plus haut) du dialogue avec les cases à cocher, le bouton Annuler est le bouton par défaut, reconnaissable par une ombre portée plus visible que sur l'autre bouton. Dans la figure 15-12 (voir plus haut) du dialogue avec les cases de choix 1 parmi N, le bouton OK est le bouton par défaut.

S'il n'y a pas de bouton par défaut et si le *focus* n'est pas sur un des boutons, l'appui de la touche Entrée ne déclenche aucun bouton.

Les éléments visuels

Ces contrôles sont totalement passifs, leur utilité est entièrement visuelle. Ils vous aideront à structurer des boîtes de dialogue complexes dans lesquelles l'utilisateur repérera facilement les éléments principaux.

La zone de groupe

Le contrôle Zone de groupe sert à encadrer plusieurs contrôles qui ont un rapport entre eux. On utilise la zone de groupe pour entourer des contrôles Case à cocher ou des contrôles Case de choix 1 parmi N. Cependant, elle peut aussi bien entourer un mélange de contrôles de différents types qui servent à définir les paramètres d'une fonctionnalité.

> ATTENTION **Un cadre, pas un conteneur**
>
> Le contrôle Zone de groupe ne contient pas les éléments qu'il encadre ; il n'a pas de rapport hiérarchique avec les autres contrôles de la boîte de dialogue.



Dans une boîte de dialogue, déposez un contrôle Zone de groupe.

Redimensionnez-le. Pour le sélectionner, cliquez sur la ligne du cadre, à gauche, à droite, en bas, mais pas en haut. Pour le déplacer, posez votre souris sur une ligne du cadre et glissez.

Si vous effacez le texte de la propriété Titre, vous obtenez un rectangle simple.

Placez plusieurs contrôles à l'intérieur du cadre de la zone de groupe. Déplacez ensuite la zone de groupe : les autres contrôles restent sur place ! En effet, la zone de groupe est seulement un élément visuel, elle n'a aucune influence sur les autres contrôles.

Les lignes horizontale et verticale



Le contrôle Ligne horizontale sépare deux régions haute et basse de la boîte de dialogue.

Vous avez des exemples dans les onglets du menu Outils > Options de OpenOffice.org. La propriété Titre de ce contrôle permet d'ajouter un texte sur la gauche de la ligne. Le contrôle Ligne verticale sépare deux régions gauche et droite de la boîte de dialogue.



Vous avez aussi des exemples dans les onglets du menu Outils > Options de OpenOffice.org. Ici en revanche, la propriété Titre n'est pas utilisée.

Les champs de saisie spécialisés

Le champ de date

Le contrôle de champ Date (`DateField`) est utilisé pour saisir une date située entre le 1er janvier 1600 et le 1er janvier 9999.



Déposez sur une boîte de dialogue un contrôle de champ Date.

Les propriétés Date, Date min, Date max permettent d'initialiser une date et de délimiter l'éventail des dates acceptées. La propriété Format de date vous offre de nombreuses variations de format, dont celle de la figure 15-13.

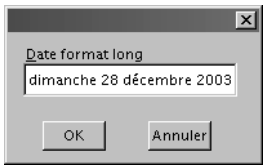


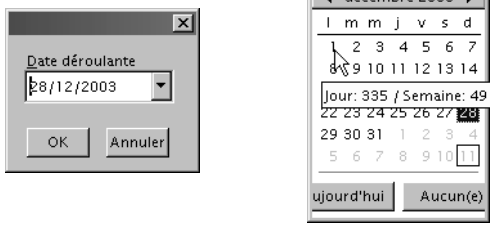
Figure 15–13
Dialogue avec un champ Date en format long

Il est à noter que l'utilisateur peut remplir ce champ avec un format différent, mais qu'il sera affiché avec le format du dialogue quand l'utilisateur passera à un autre contrôle du dialogue, par exemple en tapant la touche Tab. Parmi les formats acceptés dans la saisie, une date comme 14 juillet 75 est aussi acceptée.

La propriété Déroulante a la particularité d'afficher un petit calendrier quand l'utilisateur clique sur le triangle noir. La figure 15-14 montre l'aspect du dialogue et du calendrier.

Vous remarquerez la position du curseur sur la gauche du champ. On peut positionner le curseur par un clic de souris. Le calendrier quant à lui offre de nombreuses facilités :

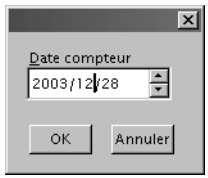
Figure 15–14
Dialogue avec un champ Date déroulant



- cliquer sur la date du mois affiché,
- changer de mois en cliquant sur les flèches droite ou gauche,
- choisir l'année précédente ou suivante en cliquant sur l'année, ce qui donne ensuite le choix du mois,
- imposer la date d'aujourd'hui.

La propriété `Compteur` permet d'incrémenter ou de décrémenter facilement le jour, le mois ou l'année, selon la position du curseur (figure 15-15).

Figure 15–15
Dialogue avec un champ Date compteur



Il est même possible de combiner l'affichage déroulant et l'affichage compteur.

Sur le plan du codage, nous devons résoudre une petite difficulté. La propriété `Date` obtenue du contrôle est un entier `Long` dont la valeur, affichée sous forme décimale, est une juxtaposition des chiffres d'année, mois, jour. Par exemple, le 28 décembre 2003 est représenté par la valeur : `20031228`. Ce format est une date ISO. Basic est capable de la convertir au format interne de type `Date` avec la fonction `CDateFromISO`.

Le codage nécessaire à la récupération de la date saisie est finalement assez simple.

```
rem Code15-07.sxw   bibli : Date Module1
Option Explicit

Sub demanderDate()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champDate As Object, dateISO As Long, laDate As Date
Dim a As Integer, m As Integer, j As Integer

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Date")
```

```
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champDate = Dlg.getControl("DateField1")
  dateISO = champDate.Date
  laDate = CDateFromISO(dateISO)
  print dateIso, laDate
end if
Dlg.Dispose
End Sub
```

Le champ horaire

Le contrôle de champ Horaire (`TimeField`) est utilisé pour saisir une heure (et minutes, secondes).

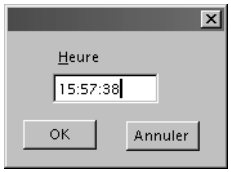


Déposez sur une boîte de dialogue un contrôle de champ horaire.

Les propriétés Heure, Heure min, Heure max permettent d'initialiser l'heure et de délimiter l'éventail des heures acceptées.

La propriété Format d'heure vous offre quelques variations de format, dont deux formats à l'anglaise. Il est à noter que l'utilisateur peut remplir ce champ avec un format différent, mais qu'il sera affiché avec le format du dialogue quand l'utilisateur passera à un autre contrôle, par exemple en appuyant sur la touche Tab. La figure 15-16 montre l'aspect du dialogue et du calendrier.

Figure 15–16
Dialogue avec un champ Heure



Vous remarquerez la position du curseur sur la droite du champ. On peut positionner le curseur par un clic de souris. La propriété Compteur permet d'incrémenter ou de décrémenter facilement l'heure, minute ou seconde, selon la position du curseur.

Sur le plan du codage, nous devons ici aussi résoudre une petite difficulté. La propriété Time obtenue du contrôle est un entier `Long` dont la valeur, affichée sous forme décimale, est une juxtaposition des chiffres d'heures, de minutes, de secondes, et d'un emplacement pour les centièmes de secondes, non utilisés. Par exemple, 15h 57min 38s est représentée par la valeur décimale : `15573800`.

Ce format n'est pas utilisable tel quel, et Basic ne nous offre pas de fonction de conversion. Nous en créons une : `ConvHeure`, qui renvoie une heure au format interne à partir de la valeur de la propriété Time.

```
rem Code15-07.sxw   bibli : Heure Module1

' conversion du champ Time en heure, minute, seconde
Function ConvHeure(champ As Long) As Date
Dim chaineHeure As String
Dim hr As Integer, mn As Integer, sec As Integer

' pour traiter les cas heure=0 et heure=minute=0
' on ajoute un nombre pour forcer l'apparition des zéros
chaineHeure = Str(champ +100000000)
hr  = CInt(Mid(chaineHeure, 3, 2))
mn  = CInt(Mid(chaineHeure, 5, 2))
sec = CInt(Mid(chaineHeure, 7, 2))
ConvHeure = TimeSerial(hr, mn, sec)
End Function
End Sub
```

Le codage nécessaire à la récupération de l'heure saisie est lui-même assez simple.

```
rem : Code15-07.sxw   bibli : Heure Module1
Option Explicit

Sub demanderHeure()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champHeure As Object, uneHeure As Date
Dim HeureBrute As Long

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Heure")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champHeure = Dlg.getControl("TimeField1")
  HeureBrute = champHeure.Time
  ' convertir le format spécial en heure interne
  uneHeure = ConvHeure(HeureBrute)
  print HeureBrute, uneHeure
end if
Dlg.Dispose
End Sub
```

Le champ monétaire



Le contrôle de champ Monétaire (`CurrencyField`) est utilisé pour saisir la valeur numérique d'une somme monétaire. Déposez sur une boîte de dialogue un contrôle de champ monétaire. En plus des propriétés classiques : Valeur, Valeur min, Valeur max, il existe une série de propriétés permettant de formater le champ affiché :

- Décimales,
- Séparateur de milliers,
- Symbole monétaire,
- Placer le symbole avant le nombre.

Saisissez un nombre de manière classique dans une des propriétés Valeur ; au passage à une autre propriété, la valeur saisie sera formatée conformément à l'ensemble des propriétés de formatage. La figure 15-17 montre un exemple de dialogue avec champ Monétaire.



Figure 15–17
Dialogue avec un champ Monétaire

Au niveau du codage, la valeur du champ monétaire est récupérée avec la propriété `Value`, qui est de type `Double` (et non pas `Currency`).

```
rem Code15-08.sxw   bibli : Monnaie
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champMonnaie As Object, Prix As Double

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Monnaie")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champMonnaie = Dlg.getControl("CurrencyField1")
  prix = champMonnaie.Value
  print Format(prix*6.55957, "FF # ### ### ##0.00")
end if
Dlg.Dispose
End Sub
```

BOGUE Fonction Format

Dans OpenOffice.org 1.1, la fonction `Format` est minée par un grand nombre de bogues (pour notre exemple, ce n'est pas important). Évitez de l'utiliser. La fonction devrait être corrigée pour la version 2.0.

Le champ masqué

Comme un masque qui ne découvre que certaines zones, le contrôle de champ Masqué (`PatternField`) est utilisé pour remplir des zones dans un champ de saisie. Dans chacune de celles-ci, l'éventail des caractères acceptés peut être limité. Par exemple, on utilisera un champ masqué pour demander un numéro de compte bancaire comprenant des chiffres et une lettre. Un exemple d'utilisation est représenté sur la figure 15-18.

Figure 15–18
Dialogue avec un champ Masqué





Déposez sur une boîte de dialogue un contrôle de champ Masqué.

La propriété Texte indique la valeur initialement affichée dans le contrôle. Le fonctionnement du contrôle masqué dépend de trois paramètres essentiels. Le Masque de saisie (`PatternField`) indique, caractère par caractère, quels ensembles de valeurs sont acceptables.

Tableau 15–1 Lettres du masque de saisie

Lettre	Valeur acceptée	Signification
L	Littéral	Reproduire le caractère correspondant du masque littéral.
N	Numérique	Seuls les chiffres 0 à 9 sont acceptés.
a	Alphabétique	Les lettres minuscules ou majuscules seules sont acceptées, y compris les accentuées.
A	Alphabétique	Comme « a », mais le caractère est affiché en majuscule.
c	Caractère	Lettre ou chiffre ; équivalent à : « a » ou « N ».
C	Caractère	Comme « c », mais le caractère est affiché en majuscule.
x	Caractère	Caractère quelconque, y compris les ponctuations, etc.
X	Caractère	Comme « x », mais le caractère est affiché en majuscule.

Le Masque littéral indique, caractère par caractère, ce qui doit être affiché par défaut dans le champ. Si le caractère correspondant du Masque de saisie vaut `L`, le caractère du Masque littéral est reproduit tel quel, et restera figé. Sinon, on va en général mettre un caractère espace ou souligné pour indiquer un emplacement à remplir.

La propriété Vérification de Format fonctionne ainsi :

- Si la valeur est `Oui`, l'affichage dans le champ est reformaté selon les autres paramètres au cours de la frappe ; la valeur récupérée par la macro correspond à ce qui est affiché.

- Si la valeur est Non, l'affichage est reformaté seulement quand le *focus* passe sur un autre contrôle ; la valeur récupérée par la macro correspond à ce qui a été tapé, non à ce qui est affiché.

Ceci sera plus clair en variant les paramètres sur un exemple. La valeur résultante du contrôle est récupérée en codage dans la propriété String, qui est du type String. Un codage typique se présente ainsi :

```
rem Code15-08.sxw   bibli : Masque
Option Explicit

Sub Main3()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
Dim champMasque As Object, saisie As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Masque")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champMasque = Dlg.getControl("PatternField1")
  saisie = champMasque.String
  print saisie
end if
Dlg.Dispose
End Sub
```

Le champ formaté

Le contrôle de champ Formaté (FormattedField) offre une grande variété de formats d'affichage, dont ceux vus dans les champs Date, Horaire et Monétaire, d'autres formats qui lui sont spécifiques, et la possibilité de créer sa propre variante de format. Évidemment, cette puissance implique une plus grande difficulté de maîtrise.

Nous allons présenter ici seulement des formats spécifiques au champ Formaté et qui sont pré-définis. Les différents exemples sont réalisés et utilisés de manière similaire, et nous expliquerons essentiellement le premier.



Sur une boîte de dialogue, déposez des boutons OK et Annuler, et l'icône Champ Formaté.

Format Date

Le panneau Propriété (figure 15-19) vous permet de définir le type de format, la valeur par défaut et les valeurs minimale et maximale.

En cliquant sur l'icône à droite du champ Formatage, vous retrouvez le panneau de dialogue du formatage de cellule dans Calc, onglet Nombre.

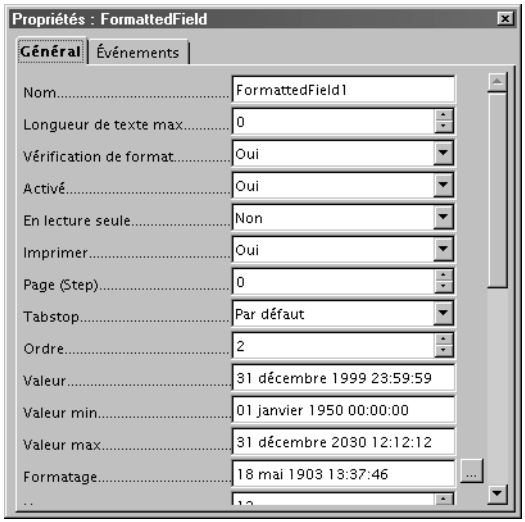


Figure 15–19
Panneau des propriétés du contrôle champ Formaté

Ici, nous avons choisi dans le format de type Date un affichage comportant la date et l'heure, avec le nom du mois dans la date. Cet affichage existe aussi dans le format de type Heure.

> **MÉTHODE Initialiser le format**
>
> Quel que soit le type de format, veillez à donner une valeur initiale à la propriété Valeur. Activez aussi la vérification de format.

Utilisez la fonction de test pour obtenir la figure 15-20.



Figure 15–20
Dialogue avec un champ Formaté en Date

Effacez tout le champ, puis remplissez-le avec `15/8/6`, et appuyez sur la touche Tab. L'affichage change pour représenter la date/heure dans le format requis. L'utilisateur peut employer divers formats de saisie, par exemple une heure sous la forme `1:2 pm` qui sera interprétée comme 13h 2min 0s ; il peut aussi modifier une partie de l'affichage. Cependant, la combinaison de la date et de l'heure dans un seul champ est en général moins compréhensible que deux champs séparés.

Au plan de la programmation, la valeur du champ est bizarrement obtenue en utilisant l'objet `Model` inclus dans l'objet champ `Formaté`. D'autre part, le type de la valeur obtenue dépend du format utilisé. Aussi utiliserons-nous dans nos exemples un type `Variant` et des fonctions de conversion quand ce sera nécessaire.

```
rem Code15-08.sxw   bibli : Formate Module1
Option Explicit

Sub FormatDate()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim ChampValeur As Object, chmod As Object, valeur As Variant

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Formate")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  ChampValeur = Dlg.getControl("FormattedField1")
  chmod = ChampValeur.Model
  valeur = chmod.EffectiveValue
  print "Date et heure : " & CDate(valeur)
  print "Min=" & CDate(chmod.EffectiveMin);
  print "  Max=" & CDate(chmod.EffectiveMax)
end if
Dlg.Dispose
End Sub
```

La valeur est récupérée sous forme de `Double` que nous devons convertir en `Date` pour l'affichage. Les valeurs extrêmes acceptées sont accessibles dans les propriétés `EffectiveMin` et `EffectiveMax` de l'objet `Model` du contrôle.

Format Pourcentage

La figure 15-21 montre comment la valeur `0,25` est affichée au format Pourcentage :

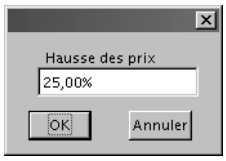


Figure 15–21
Dialogue avec un champ Formaté en Pourcentage

La programmation est identique au cas précédent. L'exemple est simplifié :

```
rem Code15-08.sxw   bibli : Formate Module2
Option Explicit
```

```
Sub FormatPourcent()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim ChampValeur As Object, valeur As Variant

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Formate")
monDialogue = bibli.GetByName("Dialog2")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  ChampValeur = Dlg.getControl("FormattedField1")
  valeur = ChampValeur.Model.EffectiveValue
  print valeur
end if
Dlg.Dispose
End Sub
```

Ici, il n'est pas nécessaire de convertir la valeur, quoique cela ne ferait aucun mal.

Format Monétaire

Nous avons choisi un format monétaire qui affiche en rouge les valeurs négatives. Évidemment, sur la figure 15-22, c'est difficile à voir.

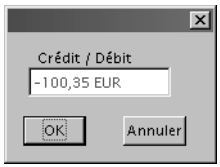


Figure 15–22
Dialogue avec un champ Formaté en Monétaire

La valeur résultante est encore un `Double`. La programmation est identique au cas précédent.

```
rem Code15-08.sxw   bibli : Formate Module3
Option Explicit

Sub FormatMonetaire()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim ChampValeur As Object, valeur As Variant

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Formate")
monDialogue = bibli.GetByName("Dialog3")
Dlg = CreateUnoDialog(monDialogue)
```

```
if Dlg.Execute = exitOK then
  ChampValeur = Dlg.getControl("FormattedField1")
  valeur = ChampValeur.Model.EffectiveValue
  print valeur
end if
Dlg.Dispose
End Sub
```

Format Booléen

Ce format affiche en clair VRAI ou FAUX selon que le contenu est True ou False, ou qu'on a mis une valeur zéro ou différente de zéro. La figure 15-23 utilise une particularité de ce format : on peut choisir la langue du texte (ici, l'indonésien).



Figure 15–23
Dialogue avec un champ Formaté en Booléen

La valeur résultante est une valeur numérique zéro ou un, que nous convertissons en un type Boolean.

```
rem Code15-08.sxw   bibli : Formate Module4
Option Explicit

Sub FormatBooleen()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim ChampValeur As Object, valeur As Variant

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Formate")
monDialogue = bibli.GetByName("Dialog4")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  ChampValeur = Dlg.getControl("FormattedField1")
  valeur = ChampValeur.Model.EffectiveValue
  print valeur, CBool(valeur)
end if
Dlg.Dispose
End Sub
```

Sur le plan visuel et ergonomique, la case à cocher ou la zone de liste simple sont bien plus utilisés que le champ Formaté Booléen.

Format Fractionnaire

Il s'agit d'une présentation utilisée dans les pays anglo-saxons. Un nombre réel est représenté par sa partie entière et une fraction correspondant *approximativement* à la partie fractionnaire. La figure 15-24 montre la représentation du nombre `3,3`.



Figure 15–24
Dialogue avec un champ Formaté en Fractionnaire

Faites le calcul :

```
3 + 2/7 = 3,285714285714285714
```

Si vous saisissez la valeur `3,3` et cliquez sur `OK`, vous obtiendrez le résultat ci-dessus, et non pas ce que vous avez saisi ! En revanche, le résultat est exact pour `5,25` et d'autres valeurs. Par une bizarrerie d'implémentation, si vous mettez `3,3` comme valeur initiale dans l'EDI et si vous acceptez cette valeur, vous récupérez bien `3,3`.

La valeur obtenue par programmation est de type `Double`. Le codage est identique à ce qui a déjà été vu.

```
rem Code15-08.sxw   bibli : Formate Module5
Option Explicit

Sub FormatFraction()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim ChampValeur As Object, valeur As Variant

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Formate")
monDialogue = bibli.GetByName("Dialog5")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  ChampValeur = Dlg.getControl("FormattedField1")
  valeur = ChampValeur.Model.EffectiveValue
  print valeur
end if
Dlg.Dispose
End Sub
```

En pratique, à cause de son manque de précision, le champ `Formaté Fractionnaire` sera plutôt limité à des affichages en lecture seule.

La sélection de fichiers

Le contrôle Sélection de fichiers (`FileControl`) est assez primitif, mais peut être utile pour rechercher facilement un fichier.



Sur une boîte de dialogue, déposez une icône Sélection de fichiers.

En augmentant suffisamment sa largeur, vous obtiendrez un dialogue similaire à celui de la figure 15-25.

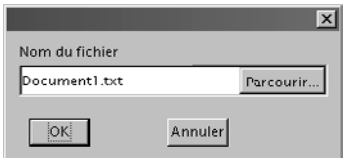


Figure 15–25
Dialogue avec un champ Sélection de fichiers

Utilisez le mode test pour voir comment fonctionne le contrôle. Cliquez sur le bouton Parcourir... pour explorer un disque de votre PC. Si vous avez inscrit un nom de fichier dans la propriété Text, constatez que ce nom sera remplacé par le nom du fichier choisi. Les noms génériques comme `*.sxw` ne sont pas interprétés comme filtre de recherche.

Par programmation, vous initialisez la propriété Text avec le chemin initial de recherche ; à la fermeture de la boîte de dialogue, vous récupérez dans la même propriété le chemin complet d'accès au fichier. Ce chemin est présenté dans le format natif du système d'exploitation.

```
rem Code15-09.sxw   bibli : Fichiers
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim champFich As Object, cheminFich As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Fichiers")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
champFich = Dlg.getControl("FileControl1")
' répertoire initial (le chemin suit la syntaxe du système)
champFich.Text = "C:\Photos\"
if Dlg.Execute = exitOK then
  cheminFich = champFich.Text
  MsgBox cheminFich
end if
Dlg.Dispose
End Sub
```

Au chapitre 16, nous verrons une autre méthode de recherche de fichiers qui offre plus de possibilités.

Le contrôle Image



Le contrôle Image, ou contrôle Picto (`ImageControl`) sert à afficher un fichier image. Les types de fichiers sont pour la plupart reconnus. Déposez sur une boîte de dialogue un contrôle Picto. Dessinez-le suffisamment grand.

Déposez ensuite seulement un bouton OK, puisqu'il s'agit d'un simple affichage. Les propriétés spécifiques au contrôle Image sont :

- Image : donne accès à une fenêtre de recherche de fichier ; une fois choisi, celui-ci s'affiche dans le contrôle.
- Échelle : avec la valeur par défaut, `Oui`, l'image est redimensionnée pour couvrir totalement la surface du contrôle, en réduisant ou augmentant la taille. Si les proportions hauteur sur largeur sont différentes de celles de l'image, l'affichage sera déformé. Avec la valeur `Non`, l'image est affichée telle quelle ; si elle est plus grande que le contrôle, une partie sera visible, si elle est plus petite, elle ne couvrira pas toute la surface du contrôle.

Choisissez donc une image, grande ou petite. Vous obtiendrez un dialogue similaire à celui de la figure 15-26.



Figure 15–26
Dialogue avec un contrôle Image

Nous allons choisir par programme le fichier qui sera affiché dans le contrôle Image. Pour cela, nous afficherons un premier dialogue pour récupérer le chemin d'accès au fichier. C'est le codage vu dans le chapitre « La sélection de fichiers ». Ensuite, nous utiliserons cette donnée pour afficher le contrôle Image dans un deuxième dialogue.

```
rem Code15-09.sxw   bibli : Fichiers
Option Explicit

Sub Main2()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim champFich As Object, cheminFich As String

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
' charger la bibliothèque en mémoire
DialogLibraries.LoadLibrary("Fichiers")
```

```
bibli = DialogLibraries.GetByName("Fichiers")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champFich = Dlg.getControl("FileControl1")
  cheminFich = champFich.Text
end if
Dlg.Dispose

Dim champImage As Object
bibli = DialogLibraries.GetByName("Images")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
champImage = Dlg.getControl("ImageControl1")
champImage.Model.ImageURL = ConvertToURL(cheminFich)
Dlg.Execute
Dlg.Dispose
End Sub
```

Dans la deuxième partie du codage, nous réutilisons la variable `Dlg` pour accéder au deuxième dialogue. Auparavant, la méthode `Dispose` a libéré les ressources du premier dialogue.

Au niveau programmation, on accède à la propriété `Image` de l'EDI avec la propriété `ImageURL` de l'objet `Model` du contrôle Image. En général, les propriétés de configuration d'un contrôle se trouvent dans la partie `Model` de celui-ci. Nous y reviendrons dans le chapitre 16.

La propriété `ImageURL` doit contenir, vous l'aviez deviné, une URL. Comme nous avons récupéré un chemin de fichier au format natif, nous utiliserons la fonction OOoBasic `ConvertToURL` de conversion au format URL.

Enfin, puisque notre deuxième dialogue est un simple affichage, il suffit d'appeler la fonction `Execute` sans utiliser l'information qu'elle renvoie.

Les barres de défilement

Les contrôles Barre de défilement (`ScrollBar`) sont horizontales ou verticales. Ces contrôles peuvent servir à créer des ascenseurs pour faire défiler le contenu d'une boîte de dialogue, mais ceci est assez complexe et rarement utile.

Nous utiliserons ces contrôles comme des glissières de réglage semblables aux dispositifs utilisés dans les amplificateurs Hi-Fi pour ajuster les graves et les aigus. On peut ainsi modifier de manière analogique (par glissement de la souris) la valeur d'une variable, par exemple commandant un niveau sonore.



Déposez sur une boîte de dialogue des boutons OK et Annuler, et une icône Barre de défilement horizontale.

Affectez à la propriété Valeur de défilement la valeur 20. Vous obtenez le dialogue de la figure 15-27.



Figure 15–27
Dialogue avec un contrôle Barre de défilement

L'utilisateur fait glisser le curseur de défilement de diverses manières :

- avec sa souris, en faisant glisser le curseur ;
- en cliquant sur les flèches aux extrémités de la barre (petit déplacement) ;
- en cliquant sur la zone de déplacement de la barre (grand déplacement) ;
- en appuyant sur les touches de déplacement droite et gauche (petit déplacement) ;
- en appuyant sur les touches de saut de page (grand déplacement).

Les valeurs de petit et grand déplacement sont initialisées dans l'EDI avec les propriétés Incrément de ligne et Incrément de bloc respectivement.

Au plan de la programmation, la valeur correspondant à la position du curseur se trouve dans la propriété `Value`, qui est du type `Double`.

```
rem Code15-11.sxw   bibli : Reglage
Option Explicit

Sub Main1()
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As Integer
Dim champReglage As Object, reglage As Double

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Reglage")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  champReglage = Dlg.getControl("ScrollBar1")
  reglage = champReglage.Value
  Print "Régler le volume à : " & reglage & "%"
end if
Dlg.Dispose
End Sub
```

La barre de progression

Le contrôle Barre de progression (`ProgressBar`) sert à présenter une graduation proportionnelle à une valeur. C'est l'équivalent d'une jauge analogique.

POUR LES EXPERTS Exécution parallèle de lignes de codes

Basic ne permet pas de gérer plusieurs *threads*. Il n'est donc pas possible d'afficher une boîte de dialogue avec une Barre de progression qui avancerait à mesure de l'avancement d'un autre processus Basic exécuté en parallèle. L'exemple que nous présentons utilise une particularité des routines de traitement d'événement sur un contrôle.

Dans une boîte de dialogue, déposez un bouton ordinaire, un bouton Annuler et un bouton OK. Pour ce dernier, mettez à Non la propriété Activé. Ajoutez enfin l'icône d'une Barre de Progression. Donnez à la propriété Valeur de progression la valeur 10. La barre de progression apparaît. Il est possible de choisir une autre couleur pour le fond du contrôle et pour les barrettes de progression. Les valeurs `Valeur de progression min` et `max` de la progression sont par défaut zéro et 100. La figure 15-28 montre l'aspect de la barre de progression.

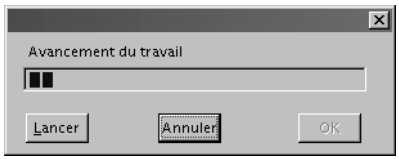


Figure 15–28
Dialogue avec un contrôle Barre de progression

La difficulté de programmation est de simuler un travail qui fait avancer peu à peu la barre de progression. Pour la surmonter, nous utiliserons les possibilités offertes par la gestion des événements. Voici la totalité du code que nous allons ensuite expliquer en détail.

```
rem Code15-10.sxw   bibli : Progres
Option Explicit

Private Dlg As Object

Sub BarreProgression()
Dim bibli As Object
Dim monDialogue As Object, exitOK As Integer

exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("Progres")
monDialogue = bibli.GetByName("Dialog1")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute <> exitOK then
  MsgBox "annulation du travail"
  ' ici il faudrait annuler ce qu'on a commencé à faire
end if
Dlg.Dispose
End Sub

Sub Avancer()
Dim av As Double
Dim ok As Object, demarre As Object, champProgres As Object
```

```
demarre = Dlg.getControl("DemarrerBouton")
demarre.Enable = False ' désactiver le bouton
champProgres = Dlg.getControl("ProgressBar1")
' récupérer la valeur initiale d'avancement
av = champProgres.Value
Do While av < 100.0
  wait 100 ' attendre 100 millisecondes
  av = av + 2
  champProgres.Value = av ' la barre s'allonge...
Loop
ok = Dlg.getControl("OKbouton")
ok.Enable = True ' activer le bouton
print "Fin de la routine Avancer()"
End Sub
```

La variable `Dlg` est déclarée comme `Private` car elle est utilisée par les deux routines du module.

Nous pourrions initialiser la valeur de la barre de progression avec la propriété `Value`, qui est du type `Double`. Ici c'est inutile car déjà fait avec l'EDI.

L'appel de la fonction `Execute` va tester si le dialogue se termine normalement ou s'il a été annulé. Dans ce cas, il est nécessaire de défaire ce qui a été fait.

La routine `Avancer` est lancée par l'événement Lors du déclenchement du bouton Lancer ; cette affectation se fait avec l'EDI, comme décrit plus haut dans la section « Premières notions d'événement ». Nous simulons un travail qui prend un certain temps avec l'instruction `wait` et une augmentation progressive de la valeur du champ de progression.

À la fin de la boucle, le travail est terminé. Le bouton OK est alors activé afin que l'utilisateur puisse éventuellement le déclencher. Une instruction `print` a été ajoutée pour visualiser cet instant.

Il est important de comprendre qu'à tout moment pendant l'exécution de la boucle, l'utilisateur peut déclencher le bouton Annuler. Alors la boîte de dialogue disparaît, mais la routine `Avancer` continue. Quand elle est terminée, l'exécution du programme reprend à l'instruction `if` de la routine `BarreProgression`.

De la même manière, l'utilisateur pourrait actionner plusieurs fois le bouton Lancer, ce qui relancerait la routine `Avancer` avec des conséquences gênantes. Pour les éviter, nous désactivons ce bouton en début de routine, en positionnant sa propriété `Enable` à `False`.

Conclusion

Nous avons passé en revue les fonctionnalités de base des contrôles et boîtes de dialogue. Les notions décrites suffisent à bien des applications, même complexes.

Pour la simplicité de l'exposé, chaque exemple utilisait un nombre minimal de contrôles. Pour vous exercer, essayez de réaliser des boîtes de dialogue comportant plusieurs contrôles et programmez la récupération des informations utilisateur. Soignez l'aspect esthétique de vos dialogues. Inspirez-vous des applications sur votre PC, en particulier OpenOffice.org : le panneau Rechercher et Remplacer, chaque onglet du menu Outils > Options, et surtout les Autopilotes. Ces derniers utilisent des notions plus avancées, mais restent basés sur les contrôles que nous avons décrits.

Au fil de vos exercices, vous constaterez certaines limitations; vous serez alors mûr pour aborder le chapitre suivant, qui vous apportera de nouvelles possibilités.

16

Créer des boîtes de dialogue élaborées

Vous avez vu dans le chapitre précédent les éléments qui constituent une boîte de dialogue et vous savez maintenant comment construire de telles boîtes simples et efficaces. Cependant, vous avez peut-être buté sur quelques limitations. Ce chapitre vous aidera à les surmonter et à concevoir des fenêtres de dialogue encore plus puissantes.

Modifier dynamiquement un contrôle

Bien qu'il soit préférable de définir à partir de l'EDI les paramètres de chacun des éléments d'une boîte de dialogue, vous aurez parfois l'information nécessaire seulement au moment de l'exécution de la macro. L'API d'OpenOffice.org offre de nombreuses propriétés et méthodes permettant de connaître l'état d'un contrôle et de le modifier.

Le contrôle et son modèle

Dans la conception logicielle des dialogues, OpenOffice.org distingue le contrôle lui-même et son modèle. *Grosso modo*, les propriétés du modèle gèrent tout ce qui est modifiable par le panneau Propriétés dans l'EDI, alors que les aspects liés à l'exécution sont traités directement par le contrôle. Le panneau de dialogue lui-même possède aussi son modèle.

> **API Concepts logiciels**
>
> Le chapitre 11.5 du « Developer's Guide » décrit techniquement les concepts d'implémentation des dialogues.

L'objet contrôle (ou dialogue) et son objet modèle sont liés. Une modification de l'un peut déclencher une mise à jour de l'autre et réciproquement. D'autre part, il arrive que la même information soit accessible via le contrôle et via son modèle, avec des noms de méthode ou propriété identiques ou différents...

Un contrôle est obtenu à partir de l'objet dialogue. Il existe deux manières d'accéder au modèle d'un contrôle. On suppose ici que la variable `Dlg` représente l'objet dialogue.

```
dim k As Object, mk As Object, mk2 As Object, md As Object
k = Dlg.getControl("TextField1")
mk = k.Model   ' le modèle du contrôle TextField1
md = Dlg.Model ' le modèle du panneau de dialogue
' deuxième manière d'accéder au modèle du contrôle TextField1
mk2 = md.getByName("TextField1")
```

Nous utiliserons la première manière, sans utiliser la variable intermédiaire `mk` pour accéder à une propriété ou méthode du modèle.

```
k = Dlg.getControl("TextField1")
k.Model.ReadOnly = True
```

L'exemple suivant montrera plusieurs exemples typiques.

Les propriétés de chaque contrôle sont nombreuses, tant au niveau du contrôle qu'au niveau de son modèle. Vous trouverez sur le zip téléchargeable, dans le répertoire consacré à ce chapitre, le fichier `PropsDialogue.sxc` qui contient la liste des propriétés utiles de chacun des contrôles.

Exemple de modification d'un dialogue

Dans l'EDI, insérez une nouvelle boîte de dialogue. Renommez son onglet `Dlg1`. Remplissez une boîte de dialogue, simplement en déposant les contrôles suivants :

- étiquette `Label1`,
- case à cocher `CheckBox1`,
- champ numérique `NumericField1`,
- zone de liste `ListBox1` (mettre Déroulante = `oui`),
- champ texte `TextField1` (mettre Plusieurs lignes = `oui`),
- bouton `CommandButton1`,
- bouton intitulé `Remplir`, que nous n'utiliserons pas dans un premier temps.

Vous obtiendrez un panneau analogue à celui de la figure 16-1. Évidemment, il ne signifie pas grand-chose, mais il a l'avantage d'employer des propriétés couramment utilisées, et qui se retrouvent sur la plupart des contrôles. L'exécution de la macro que nous allons écrire changera l'aspect du panneau pour le rendre conforme à celui de la figure 16-2.

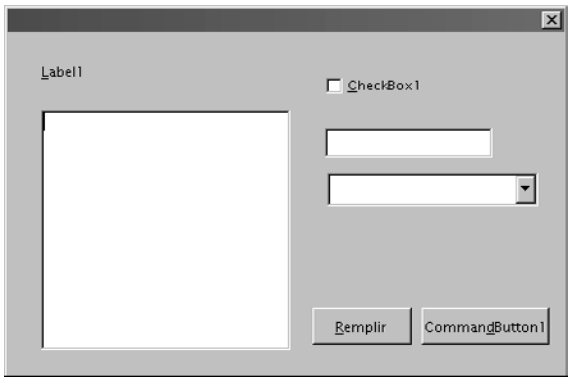


Figure 16–1
Panneau de dialogue « brut »

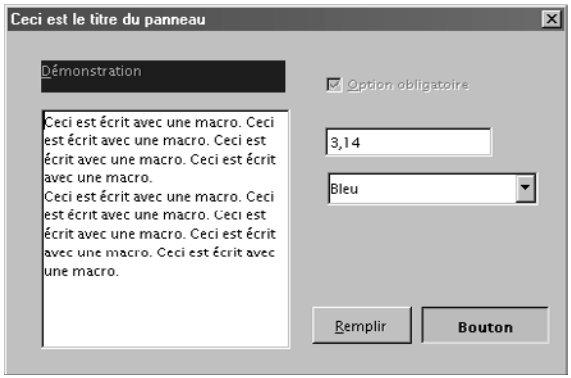


Figure 16–2
Panneau de dialogue modifié par macro

Voici le codage, nous allons le commenter ensuite :

```
rem Code16-01.sxw  bibli : ModifDyn Module1
Option Explicit

Private Dlg As Object

Sub DialogueDynamique()
Dim bibli As Object, monDialogue As Object

bibli = DialogLibraries.GetByName("ModifDyn")
monDialogue = bibli.GetByName("Dlg1")
Dlg = CreateUnoDialog(monDialogue)
ModifierDialogue
```

```
Dlg.Execute
Dlg.Dispose
End Sub

Sub ModifierDialogue()
Dim k As Object
Dlg.Model.Title = "Ceci est le titre du panneau"

k = Dlg.getControl("Label1")
k.Model.Label = "Démonstration"
k.Model.TextColor = RGB(255,150,0)    ' orange
k.Model.BackgroundColor = RGB(0,50,0) ' vert sombre

k = Dlg.getControl("CheckBox1")
k.Model.Label = "Option obligatoire"
k.Model.State = 1 ' cocher la case
k.Model.Enabled = False ' inhiber le contrôle

k = Dlg.getControl("NumericField1")
k.Model.ValueMin = 1
k.Model.ValueMax = 999.9999
k.Model.Value = pi ' valeur initiale (nombre Pi)

k = Dlg.getControl("CommandButton1")
k.Model.Label = "Bouton"
k.Model.FontWeight = com.sun.star.awt.FontWeight.BOLD
k.Model.BackgroundColor = RGB(100,255,255)' cyan clair
k.Model.State = 1 ' enfoncer le bouton

k = Dlg.getControl("TextField1")
Const texte = "Ceci est écrit avec une macro. "
k.Model.Text = texte & texte & texte & texte & chr(13) & _
    texte & texte & texte & texte & texte
k.Model.ReadOnly = True

Dim tableChoix() As Variant
tableChoix = Array("Noir","Vert","Bleu","Rouge","Jaune","Blanc")
k = Dlg.getControl("ListBox1")
k.Model.StringItemList() = tableChoix()
k.addItem("Violet",1) ' rajouter un choix en position 1
k.selectItemPos(3, true) ' sélectionnera le choix Bleu
End Sub
```

Le sous-programme principal `DialogueDynamique` crée le dialogue, appelle le sous-programme `ModifierDialogue`, affiche le dialogue et libère la ressource quand le dialogue est fermé. Nous ne nous occupons pas de la valeur de retour du dialogue. En fait, le seul moyen de le fermer sera de cliquer sur la case X de fermeture de la fenêtre.

La variable `Dlg` est définie à l'extérieur du sous-programme principal afin de l'utiliser dans le sous-programme appelé. Nous en expliquerons la raison plus tard.

Le sous-programme `ModifierDialogue` va complètement changer l'aspect de la boîte de dialogue. L'ordre des changements effectués est sans importance.

Nous commençons par donner un titre dans le bandeau de la fenêtre de dialogue. Puis, nous allons tour à tour traiter les contrôles, en affectant chacun d'eux à la variable `k`. Le texte du contrôle étiquette est modifié, ainsi que la couleur de la police et la couleur du fond. Le texte de la case à cocher est modifié, cette case est cochée (la valeur zéro enlèverait la coche) et nous empêchons l'utilisateur de l'utiliser en la mettant en état inhibé. L'utilisateur verra un contrôle grisé. Pour le champ numérique, nous changeons les valeurs minimale et maximale admissibles et donnons une valeur initiale.

Nous changeons le texte du premier bouton, puis nous le mettons en gras. Les valeurs de `Model.FontWeight` sont décrites dans le chapitre 11 consacré à Writer, ainsi que d'autres propriétés de contrôles dont le nom commence par `Font`. Le fond devient couleur bleu-vert clair. Les propriétés de format de police existent pour tous les contrôles qui contiennent des caractères.

En mettant à 1 la propriété `Model.State` le bouton va paraître enfoncé. La valeur normale est zéro.

Le champ texte nous sert à afficher un texte sur plusieurs lignes. Notez que le formatage (police, couleur, taille, etc.) s'applique au texte tout entier. Le contrôle est ensuite mis en lecture seule, ce qui empêche toute modification du contenu par l'utilisateur. Cette propriété existe dans les autres contrôles.

Finalement, nous remplissons les choix de la zone de liste.

Programmation du contrôle zone de liste

Les différentes valeurs de la liste de choix sont remplies à partir d'un tableau de chaînes de caractères. Pour simplifier, nous avons initialisé ce tableau avec la fonction `Array` de Basic. Remarquez l'usage des parenthèses vides pour préciser que tout le tableau est concerné dans l'affectation.

Une autre méthode courante consiste à récupérer la liste de choix existante pour obtenir un tableau de chaînes de caractères, modifier celui-ci et le réaffecter à la liste de choix.

```
tableChoix() = k.Model.StringItemList()
' - - modifier tableChoix() - -
k.Model.StringItemList() = tableChoix()
```

Nous utilisons la méthode `Model.addItem` de l'objet zone de liste pour insérer un choix supplémentaire dans la liste ; ses deux arguments sont, dans l'ordre :

1. l'intitulé de l'élément sous forme d'une chaîne de caractères,
2. la position d'insertion de cet élément.

Notez que les positions d'éléments de choix sont numérotées à partir de zéro et sont des valeurs `Integer`.

La méthode `Model.addItems` (avec un « s » final) insère plusieurs éléments ; ses deux arguments sont respectivement :

1. un tableau de chaînes de caractères,
2. la position d'insertion du premier élément.

> **À NOTER Performance**
>
> Lors d'ajouts en nombre d'éléments à une liste déroulante, préférez l'utilisation de la méthode `addItems`, quitte à passer par un tableau intermédiaire. Les appels d'insertion d'éléments sont coûteux et insérer les éléments un à un avec la méthode `addItem` risque de dégrader les performances de votre macro jusqu'à la rendre inutilisable.

Nous pourrions enlever des éléments grâce à la méthode `Model.removeItems`, qui utilise aussi deux arguments :

1. la position du premier élément à supprimer,
2. le nombre d'éléments consécutifs à supprimer.

Nous sélectionnons ensuite un des éléments afin qu'il soit affiché. Nous utilisons pour cela la méthode `Model.selectItemPos`, à deux arguments :

1. la position de l'élément à sélectionner,
2. un indicateur booléen mis à `True` pour sélectionner, ou `False` pour désélectionner.

Nous aurions pu utiliser la méthode `Model.selectItem` qui a pour arguments :

1. l'intitulé de l'élément à sélectionner (valeur `String`),
2. un indicateur booléen pour sélectionner ou désélectionner.

Dans le cas d'une zone de liste où plusieurs choix simultanés sont possibles, l'indicateur booléen ci-dessus est parfaitement justifié. Dans le cas plus courant d'un choix 1 parmi N, il suffit de sélectionner un élément pour désélectionner le seul autre élément qui pourrait être sélectionné. Bien que ce soit peu courant, il est tout à fait possible d'afficher un choix sans aucun élément sélectionné.

Dans une zone de liste à plusieurs choix simultanés, la méthode `Model.selectItemsPos` permet de (dé)sélectionner plusieurs éléments en une fois. Les arguments sont :

1. un tableau indexé à partir de zéro, contenant les positions des éléments concernés,
2. un indicateur booléen pour sélectionner ou désélectionner l'ensemble.

L'utilisateur va évidemment modifier le choix proposé. À la fermeture du dialogue ou dans un traitement d'événement, vous déterminerez son choix grâce à quatre fonctions de l'objet zone de liste lui-même (pas son modèle).

Deux propriétés en lecture seule (en fait, des fonctions) sont utilisées dans les zones de liste en choix 1 parmi N :

- `SelectedItemPos`, de type `Integer`, donne la position de l'élément sélectionné.
- `SelectedItem` , de type `String`, donne l'intitulé de l'élément sélectionné.

Deux autres fonctions sont prévues pour les zones de liste à choix multiple. Elles peuvent aussi être utilisées pour un choix 1 parmi N. Attention à leurs noms, presque identiques aux deux précédents :

- `SelectedItemsPos` renvoie les positions d'éléments sélectionnés, sous la forme d'un tableau de valeurs `Integer`.
- `SelectedItems` renvoie les intitulés des éléments sélectionnés, sous la forme d'un tableau de valeurs `String`.

Comme d'habitude, les fonctions Basic `LBound` et `UBound` vous permettront de balayer ces tableaux.

La propriété `ItemCount`, en lecture seule, donne le nombre d'éléments de la liste de choix.

Mettre le focus sur un contrôle

Après un changement du `step` d'un dialogue à pages multiples (voir plus loin), ou après un traitement d'événement de dialogue, il est parfois utile de positionner le focus sur le contrôle de dialogue qui devrait logiquement être modifié. La méthode est simple :

```
monControl.setFocus
```

Remettre à « vide » un champ numérique

Comment ne mettre aucune valeur dans le champ d'un contrôle numérique ? Avec l'EDI, c'est possible en supprimant toute valeur. Dynamiquement (sur un événement), il suffit de mettre une chaîne de caractères nulle dans la propriété `Text` du contrôle.

```
champPoids.Text = ""
```

Une autre méthode, qui fonctionne aussi sur d'autres contrôles est de réinitialiser la propriété `Value` dans le modèle du contrôle :

```
champPoids.Model.PropertyToDefault = "Value"
```

Attention : la `ValueMin` ne sera pas prise en compte si l'utilisateur ne remplit pas le champ ; dans ce cas la valeur récupérée est zéro.

Traiter des événements dans un dialogue

Nous allons reprendre l'exemple en modifiant l'aspect du dialogue, non pas dès l'affichage, mais seulement quand l'utilisateur appuiera sur le bouton Remplir.

Dans le codage précédent, supprimez la ligne `ModifierDialogue` située juste avant la ligne `Dlg.Execute` (ou mettez-la en commentaire). Ne supprimez pas le sous-programme `ModifierDialogue`.

Dans l'EDI, en utilisant l'onglet Événements du panneau Propriétés du bouton Remplir, cliquez sur la case [..] de la ligne Lors du déclenchement. Utilisez le panneau qui s'ouvre pour affecter à cet événement la macro `ModifierDialogue`. La manipulation a déjà été décrite dans le chapitre 15. Si vous ne trouvez pas le nom du document contenant la macro, fermez le document et rouvrez-le. N'oubliez pas de cliquer le bouton Assigner avant de fermer par le bouton OK.

> **REMARQUE Le panneau de dialogue**
>
> Si vous sélectionnez dans l'EDI le panneau de dialogue lui-même (cliquez sur un bord), vous verrez qu'il comporte lui aussi des événements, que l'on peut gérer comme ceux des contrôles.

Exécutez la macro principale : le dialogue s'affiche, tel qu'il est défini par l'EDI (figure 16-1). Déclenchez par clavier ou cliquez sur le bouton Remplir : le contenu du dialogue change (figure 16-2).

Maintenant, vous pouvez comprendre pourquoi nous avons déclaré `Private` la variable `Dlg`. Alors que dans l'exemple initial nous aurions pu transmettre l'objet dialogue dans un argument de la macro `ModifierDialogue`, ce n'est pas possible si cette macro est déclenchée par un événement.

Dans un sous-programme déclenché sur un événement, le moyen habituel d'accéder au dialogue est d'utiliser une variable commune (déclarée `Private`, ou `Public`, ou `Global`) initialisée au préalable.

Bien que cela ne change rien dans notre exemple, le sous-programme sera lancé à chaque clic sur le bouton Remplir. Le programmeur doit en tenir compte, par exemple en positionnant un indicateur pour ne pas effectuer le traitement plus d'une fois.

Le sous-programme traitant l'événement peut aussi décider de fermer la boîte de dialogue, avec l'instruction :

```
Dlg.endExecute
```

Notez que le sous-programme continuera néanmoins à se dérouler jusqu'à sa fin. La boîte de dialogue se fermera après. La méthode `Execute` de l'objet dialogue renverra le même

résultat que si l'utilisateur l'avait fermée avec un bouton Annuler ou en fermant la fenêtre de dialogue. Le programmeur fera éventuellement la différence en analysant ses données.

> **PIÈGE Événements multiples**
>
> Sur un même contrôle, vous pouvez affecter des traitements à plusieurs des événements disponibles. Vous risquez alors de déclencher plusieurs événements pour la même action de l'utilisateur, donc des appels successifs de vos gestionnaires d'événements. Les interactions logicielles entre ces gestionnaires peuvent être très difficiles à maîtriser. Dans la mesure du possible, n'utilisez qu'un seul événement sur un contrôle.
>
> Avec les événements, vous abordez le domaine du *temps réel*, qui peut provoquer des comportements aléatoires ; par exemple, un utilisateur rapide pourra déclencher un deuxième événement pendant le temps de traitement du premier. Écrivez des gestionnaires simples et rapides, ou bloquez l'interface utilisateur pendant le traitement (voir chapitre 19).

Gestionnaire d'événement commun à plusieurs contrôles

Certains dialogues comportent plusieurs contrôles dont le traitement doit être similaire. Il est tout à fait possible d'affecter le même gestionnaire d'événement à des événements de plusieurs contrôles. Alors comment déterminer le contrôle ayant déclenché l'événement ?

Un gestionnaire d'événement peut être déclaré avec un paramètre en argument.

```
Sub GestionnaireMultiple(Evenement As Object)
```

La variable de ce paramètre est un objet événement qui peut fournir diverses informations complémentaires que nous détaillerons plus loin. Pour ce qui nous occupe, cet objet expose la propriété `Source`, qui est l'objet contrôle ayant déclenché l'événement. Disposant du contrôle, nous pouvons accéder à toutes ses propriétés, en particulier `Model.Name`, de type `String`, qui est le nom donné au contrôle, et la propriété `Model.Tag`, de type `String`, qui est désignée dans le panneau Propriétés de l'EDI sous le nom de Complément d'information. Nous pouvons initialiser ce `Tag` avec une valeur quelconque, aussi bien un nombre (en chaîne de caractères) qu'un texte constitué de plusieurs données concaténées. Un usage judicieux du `Tag` peut simplifier considérablement le codage d'un gestionnaire de multiples contrôles.

Créez une nouvelle boîte de dialogue, dont l'onglet aura pour nom `Dlg2`. Déposez sur le dialogue six boutons constitués ainsi :

- un bouton OK (type de bouton OK),
- un bouton Annuler (type Annuler),
- un bouton Fin (type Normal) nommé `BtnFini`,
- trois boutons de type Normal.

Sur les 4 boutons de type Normal, affectez l'événement Lors du déclenchement à la routine `GestionnaireMultiple` et donnez une valeur particulière à chaque propriété Complément

d'information. Ajoutez un contrôle Étiquette nommé `Affichage`, pour obtenir le panneau de la figure 16-3.

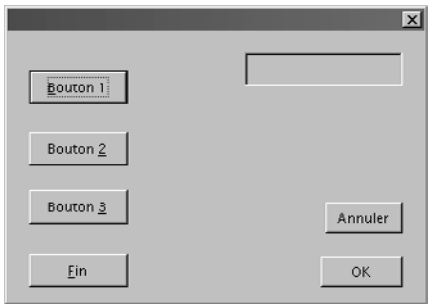


Figure 16–3
Panneau de dialogue Dlg2

Nous utiliserons le codage suivant :

```
rem Code16-01.sxw  bibli : ModifDyn Module2
Option Explicit

Private Dlg2 As Object, finSpeciale As Boolean

Sub GererPlusieursControles()
Dim bibli As Object, monDialogue As Object, resu As Integer
bibli = DialogLibraries.GetByName("ModifDyn")
monDialogue = bibli.GetByName("Dlg2")
Dlg2 = CreateUnoDialog(monDialogue)
resu = Dlg2.Execute
Dlg2.Dispose ' il vaut mieux libérer la ressource ici

if resu = com.sun.star.ui.dialogs.ExecutableDialogResults.CANCEL then
  if finSpeciale then
    MsgBox("Sortie par le bouton Fin")
  else
    MsgBox("Sortie Annuler")
  end if
else
  MsgBox("Sortie OK")
end if
End Sub

' déclenché par les boutons 1, 2, 3 et le bouton Fin
Sub GestionnaireMultiple(Evenement As Object)
Dim k As Object

' afficher le texte du complément d'information du contrôle
k = Dlg2.getControl("Affichage")
k.Model.Label = Evenement.Source.Model.Tag
' terminer le dialogue pour un bouton particulier
finSpeciale = (Evenement.Source.Model.Name = "BtnFini")
if finSpeciale then
```

```
  Wait(600) ' temporisation pour voir changer l'affichage
  Dlg2.endExecute
end if
End Sub
```

Étudions d'abord le codage de la macro `GestionnaireMultiple`. Pour tout déclenchement, elle va recopier dans le contrôle Étiquette la valeur du `Tag` du contrôle à l'origine de l'événement. Cela nous permet de vérifier l'activation du bouton par simple visualisation sur l'étiquette. Le dialogue restera ouvert, sauf si le contrôle à l'origine de l'événement a pour nom `BtnFini`. Dans ce cas, la variable `finSpeciale` prendra la valeur `True` et le dialogue sera fermé grâce à `endExecute`.

Dans le sous-programme principal, au retour de la méthode `Execute`, une analyse du résultat est effectuée. Comme la variable `finSpeciale` est commune, sa valeur nous permet de distinguer une sortie Annuler d'une sortie par le bouton Fin.

L'objet `Source` obtenu à partir de l'objet événement expose une autre propriété intéressante : `Context`. Celle-ci représente le dialogue lui-même ! Nous avons donc une autre manière de récupérer l'objet dialogue sans passer par une variable commune, ce qui donne une programmation plus élégante :

```
Sub GestionnaireMultiple(Evenement As Object)
Dim monDialogue As Object
monDialogue = Evenement.Source.Context
```

Les informations fournies par un événement

La variable `Evenement` obtenue en argument d'un sous-programme de gestion d'événement est une donnée structurée. La composition de cette structure dépend du type d'événement survenu ; seul l'élément `Source`, déjà vu, existe pour tous les événements.

La description des différentes structures d'événement se trouve dans la description IDL de la documentation SDK, dans la page sur le module `com.sun.star.awt`. À partir de la section « Struct » de cette page, suivez les liens concernant `ActionEvent`, `FocusEvent`, `KeyEvent`, `MouseEvent`, etc.

Nous allons passer en revue les principaux événements. Ne perdez pas de vue que certains événements peuvent être déclenchés par une action sur la souris ou sur le clavier, et parfois de manière indirecte.

Lors du déclenchement

Circonstances d'apparition :

- Le bouton a été déclenché (par clavier ou souris) ; seul un bouton de type Standard peut déclencher cet événement.

- Le choix a été effectué dans une zone de liste (par clavier ou souris).
- La case de choix 1 parmi N est choisie (même si on clique dessus alors qu'elle était déjà dans l'état sélectionné).
- La case à cocher a changé d'état.

Un seul élément spécifique : `ActionCommand`, de type `String`. Il contient la valeur que le programmeur a éventuellement donnée à la propriété `ActionCommand` de l'objet bouton, case de choix, case à cocher.

Changement de focus

Il s'agit des événements Réception de focus et Perte de focus.

Circonstance d'apparition : toute action qui déplace le *focus* dans le sens correspondant à l'événement. Cela peut se produire à l'occasion d'un clic ou d'un appui sur la touche Tab ou sur une touche déclenchant un autre bouton, etc.

L'élément `FocusFlags`, de type `Integer`, fournit la raison du changement de *focus*, sous la forme de la somme de constantes nommées de la forme :

```
com.sun.star.awt.FocusChangeReason.TAB
```

Chacune de ces constantes, listées au tableau 16-1, est une puissance de deux, ce qui permet de détecter sa présence avec un ET logique (opérateur `and`).

Tableau 16–1 Constantes de FocusFlags

Constante	Raison
TAB	Utilisation de la touche Tabulation.
CURSOR	Utilisation d'une touche haut, bas, page précédente, page suivante...
MNEMONIC	Utilisation d'une touche de raccourci.
FORWARD	Le *focus* est passé au contrôle suivant.
BACKWARD	Le *focus* est passé au contrôle précédent.
AROUND	Bouclage sur la liste des contrôles.
UNIQUEMNEMONIC	Le raccourci employé désigne un unique contrôle.

L'élément `Temporary`, de type `Boolean`, vaut `True` si le changement de *focus* est une conséquence indirecte d'une autre action (par exemple la fermeture de la fenêtre de dialogue).

L'élément `NextFocus` fournit la fenêtre qui reçoit le *focus* (en cas de Perte de focus).

Touche du clavier

Il s'agit des événements Touche enfoncée et Après avoir lâché la touche.

L'élément `KeyCode`, de type `Integer`, identifie la touche appuyée sous la forme d'une constante nommée (tableau 16-2), de la forme :

```
com.sun.star.awt.Key.HOME
```

L'action simultanée d'une touche Maj, Ctrl ou Alt, n'a pas d'influence sur la valeur donnée par `KeyCode`.

Tableau 16–2 Constantes de KeyCode

Constante	Touche	Constante	Touche
NUM0 à NUM9	1 à 9 (clavier numérique ou principal)	DELETE	Suppr
A à Z	lettre majuscule ou minuscule	ADD	+
F1 à F26	Touche de fonction	SUBSTRACT	-
DOWN	Direction bas	MULTIPLY	*
UP	Direction haut	DIVIDE	/
LEFT	Direction gauche	POINT	Point
RIGHT	Direction droite	COMMA	Virgule
HOME	Direction coin	LESS	<
END	Fin	GREATER	>
PAGEUP	Page précédente	EQUAL	signe =
PAGEDOWN	Page suivante	INSERT	Inser
RETURN	Entrée	SPACE	Espace
ESCAPE	Echap	BACKSPACE	Retour arrière
TAB	Tabulation		

L'élément `KeyChar`, de type `String`, donne :

- soit le caractère correspondant, en tenant compte de la touche `Maj` ;
- soit un pseudo-caractère dont la valeur `ASC()` est inférieure à 32, (exemple `Entrée` donne 13) ;
- soit un pseudo-caractère dont la valeur `ASC()` est zéro.

L'élément `KeyFunc`, de type `Integer`, indique la signification fonctionnelle de la touche, quand elle en a une. C'est une constante nommée (voir tableau 16-3) de la forme :

```
com.sun.star.awt.KeyFunction.DONTKNOW
```

Une touche ordinaire renvoie la valeur `DONTKNOW`. La touche Ctrl+C renvoie `COPY`. Certains valeurs sont particulières à certains claviers et systèmes.

Tableau 16–3 Constantes de KeyFunc

Constante	Signification	Constante	Signification
DONTKNOW	Aucune	CUT	Couper
NEW	Nouveau	COPY	Copier
OPEN	Ouvrir	PASTE	Coller
SAVE	Enregistrer	UNDO	Défaire
SAVEAS	Enregistrer sous	REDO	Refaire
PRINT	Imprimer	REPEAT	Répéter
CLOSE	Fermer	FIND	Chercher
QUIT	Quitter	FINDBACKWARD	Chercher en sens inverse
PROPERTIES	Propriétés	FRONT	Avant-plan

L'élément `Modifiers`, de type `Integer`, indique si une touche Maj, Ctrl ou Alt est pressée. À chacune correspond une constante nommée :

```
com.sun.star.awt.KeyModifier.MOD1  ' touche Ctrl
com.sun.star.awt.KeyModifier.MOD2  ' touche Alt
com.sun.star.awt.KeyModifier.SHIFT ' touche Maj
```

Les valeurs des constantes étant des puissances de deux, on détermine lesquelles sont activées avec un ET logique.

Souris

L'élément `KeyModifiers` est disponible, il a été décrit plus haut. L'élément `Buttons`, de type `Integer`, indique quels boutons de souris sont pressés. À chacun correspond une constante nommée :

```
com.sun.star.awt.MouseButton.LEFT   ' bouton gauche
com.sun.star.awt.MouseButton.RIGHT  ' bouton droit
com.sun.star.awt.MouseButton.MIDDLE ' bouton du milieu
```

> Attention, certains événements de souris peuvent se déclencher un grand nombre de fois par seconde.

Les éléments `X` et `Y`, de type `Long`, indiquent la position de la souris, exprimée en pixels, la valeur 0 correspondant respectivement au bord gauche et au bord haut de la surface du contrôle. L'élément `ClickCount`, de type `Long`, compte le nombre de clics effectués. Elle est remise automatiquement à zéro après une temporisation de quelques dixièmes de secondes. Plusieurs clics successifs déclencheront autant de fois l'événement Bouton de souris enfoncé (par exemple). La valeur ne peut pas être modifiée par programmation.

L'élément `PopupTrigger`, de type `Boolean`, indique `True` si l'événement correspond à l'ouverture d'un menu contextuel.

Texte Modifié

Cet événement apparaît dans les contrôles où l'utilisateur peut modifier un texte ou une valeur numérique. Il n'existe aucun élément spécifique de cet événement.

Dialogues à pages multiples

Les dialogues OpenOffice.org ne disposent pas de contrôle à onglets. La méthode du `Step` est ce qui s'en rapproche le plus. Ce mécanisme a été inventé pour « faciliter » l'écriture de dialogues type AutoPilote, où l'utilisateur est guidé pas à pas dans un processus à plusieurs étapes. Il utilise la propriété `Step`, appelée `Page` dans le panneau `Propriétés` de l'EDI. Cette propriété est disponible dans le panneau de dialogue lui-même et dans chacun des contrôles. On y accède par programmation sous la forme `Model.Step` ; elle est du type `Long`. Le principe consiste à rendre visible ou non chaque contrôle selon la valeur de son `Step`.

- Si le `Step` du contrôle vaut zéro, le contrôle est tout le temps visible.
- Si le `Step` du panneau vaut zéro, tous les contrôles sont visibles.
- Si le `Step` du panneau vaut N, seuls les contrôles dont le `Step` vaut N sont visibles, ainsi que ceux dont le `Step` vaut zéro.
- Par défaut, tous les `Step` sont à zéro et tous les contrôles sont visibles.

Attention : l'affichage des contrôles dans l'EDI tient compte des valeurs de `Step`, donc vous ne verrez peut-être pas tous les contrôles si le `Step` du panneau est différent de zéro.

La conception d'un tel dialogue peut amener à définir plusieurs contrôles (par exemple des boutons) superposés, car normalement un seul à la fois est visible. Dans ce cas, dans l'EDI vous devrez changer la valeur de `Step` du panneau pour accéder à un contrôle caché.

En général, on définit un ou plusieurs contrôles (boutons par exemple) ayant un `Step` nul afin qu'ils soient toujours visibles à l'exécution. Quand l'utilisateur activera un de ces contrôles, un traitement sera effectué, puis le `Step` du panneau sera modifié. La conception et la mise au point d'un dialogue à pages multiples peut être assez complexe.

Dialogues emboîtés

Position et dimensions

La position et les dimensions d'un panneau de dialogue ou d'un contrôle sont exposés par l'objet `Model` de ceux-ci.

- `PositionX` et `PositionY`, de type `Long`, sont les coordonnées du coin haut-gauche.
- `Width` et `Height`, de type `Long`, sont la largeur et la hauteur du panneau ou du contrôle.

Ces propriétés reflètent ce qui est affiché sur le panneau Propriétés de l'EDI. L'unité de mesure n'est pas le pixel, ni même le *Twips*, mais le *Map AppFont*, qui est lié aux hauteur et largeur moyenne des caractères de la police système. L'avantage est d'obtenir un panneau proportionné aux textes affichés par le système.

L'origine des coordonnées est le coin haut-gauche de la fenêtre parente. Habituellement, cette fenêtre est celle du document ouvert. Cependant, quand un dialogue est en cours, il est possible d'en ouvrir un autre, à l'occasion d'un événement intercepté par une macro. Dans ce cas, le deuxième dialogue est positionné par rapport au premier, qui est sa fenêtre parente.

Comme l'origine des coordonnées n'est pas un point fixe de l'écran, un panneau de dialogue apparaîtra à un endroit dépendant de la position de sa fenêtre parente et de la position du panneau de dialogue dans l'EDI (ce qui définit ses coordonnées X et Y). Il n'est pas possible d'avoir une maîtrise totale sur la position d'un dialogue, mais on peut éviter des effets peu ergonomiques. Selon les circonstances, on souhaitera, soit centrer le dialogue dans la fenêtre parente, soit le placer à une position relative particulière. Notez qu'il est parfaitement possible d'avoir des coordonnées négatives pour le placer à gauche ou au-dessus de la fenêtre de référence.

L'EDI permet de positionner un dialogue avec des coordonnées négatives, mais le dialogue est alors partiellement caché dans la fenêtre de l'EDI. Il est plus pratique d'imposer par programmation des coordonnées négatives, avant d'afficher le dialogue.

Un dialogue dans un dialogue

Notre exemple va vous montrer comment emboîter des dialogues et les positionner. Il est préférable d'ouvrir le document du zip téléchargeable afin de profiter des dialogues déjà configurés.

```
rem Code16-01.sxw  bibli : DialDial Module1
Option Explicit
```

```
Private Dlg(4) As Object

Sub DialoguesImbriques()
Dlg(1) = Dialogue("DialDial", "Dlg1")
if MsgBox("Centrer ?", 4) = 6 then
  CenterDialog(Dlg(1))
end if
Dlg(1).Execute
Dlg(1).Dispose
End Sub

' déclenché par les boutons Lancer Dialogue n  en position
Sub LancerDialogue(Evt As Object)
Dim n As Integer
n = Evt.Source.Model.Tag ' conversion string -> nombre
Dlg(n) = Dialogue("DialDial", "Dlg" & n) ' conv n -> String
Dlg(n).Execute
Dlg(n).Dispose
End Sub

' déclenché par les boutons Lancer Dialogue n Centré
Sub LancerDialogueCentre(Evt As Object)
Dim n As Integer
n = Evt.Source.Model.Tag ' conversion string -> nombre
Dlg(n) = Dialogue("DialDial", "Dlg" & n) ' conv n -> String
CenterDialog(Dlg(n), Dlg(n-1))
Dlg(n).Execute
Dlg(n).Dispose
End Sub

' crée un dialogue et renvoie son objet (ou Null)
Function Dialogue(nomBibli As String, nomDialogue As String) As Object
Dim conteneur As Object, bibli as Object
Dim boiteDialogue as Object
if DialogLibraries.hasByName(nomBibli) then
  conteneur = DialogLibraries
else
  if GlobalScope.DialogLibraries.hasByName(nomBibli) then
    conteneur = GlobalScope.DialogLibraries
  else
    Exit Function ' bibliothèque pas trouvée
  end if
end if
conteneur.LoadLibrary(nomBibli)
bibli = conteneur.GetByName(nomBibli)
if not bibli.hasByName(nomDialogue) then Exit Function
boiteDialogue = bibli.GetByName(nomDialogue)
Dialogue = CreateUnoDialog(boiteDialogue)
End Function
```

```
Sub CenterDialog(nouvDial As Object, optional parentDial As Object)
Dim parentSize As Object, XPos As Long, YPos As Long
' les tailles et positions sont mesurées en pixels
if isMissing(parentDial) then
  if IsNull(StarDesktop.CurrentFrame) then exit sub
  parentSize = StarDesktop.CurrentFrame.ComponentWindow.Size
else
  parentSize = parentDial.Size
end if
XPos = (parentSize.Width/2) - (nouvDial.Size.Width/2)
YPos = (parentSize.Height/2) - (nouvDial.Size.Height/2)
nouvDial.setPosSize(XPos, YPos, 0, 0, com.sun.star.awt.PosSize.POS)

End Sub
```

L'exemple utilisera, pour des raisons de programmation, un tableau d'objets destinés à mémoriser les dialogues successifs. L'index zéro ne sera pas utilisé.

CONSEIL Pour vos dialogues

Réutilisez les deux macros utilitaires `Dialogue` et `CenterDialog` dans vos dialogues, elles sont conçues dans ce but.

Le premier dialogue est obtenu grâce à la fonction utilitaire `Dialogue` que nous avons développée pour simplifier ce travail. Elle récupère la bibliothèque en argument, qu'elle soit dans le même conteneur ou dans le conteneur `soffice`. Puis, elle en extrait la boîte de dialogue et instancie le dialogue. Si la recherche est infructueuse, la fonction renvoie une valeur `Null`.

Avant d'afficher le dialogue, nous demandons si l'utilisateur veut centrer le dialogue (par rapport à la fenêtre parente). Dans l'affirmative, nous appelons une deuxième routine utilitaire `CenterDialog`.

Cette routine comporte en premier argument le nouveau dialogue à positionner. Le deuxième argument, s'il existe, est le dialogue parent (il n'existe pas de moyen de le déterminer par programmation). S'il n'y a qu'un seul argument, ce qui est notre cas ici, nous essayons de récupérer la fenêtre actuellement affichée. S'il n'y en a pas (la macro principale est lancée au démarrage d'OpenOffice.org), alors nous ne pouvons centrer le dialogue. Notez que dans une routine aussi générale, on ne peut utiliser `ThisComponent` car, si la macro principale est lancée sans document (c'est possible, avec une macro dans `soffice`), cet objet n'existe pas.

À partir de l'objet fenêtre, nous obtenons ses dimensions (mesurées en pixels) avec sa propriété `Size`. Pour simplifier le codage, nous allons aussi utiliser la propriété `Size` du nouveau dialogue, qui elle aussi est mesurée en pixels. Les coordonnées du coin du dialogue sont calculées pour le positionner au centre, puis nous utilisons la méthode `setPosSize` pour imposer la nouvelle position. La méthode `setPosSize` peut imposer tout ou partie

des coordonnées ou dimensions. Ici, nous imposons les coordonnées X et Y ; les paramètres de rangs 3 et 4 (largeur, hauteur) ne sont pas utilisés.

> Remarquez que la méthode `getPosSize` (ou la pseudo-propriété `PosSize`, en lecture seule) renvoie une structure de rectangle (`X`, `Y`, `Width`, `Height`), mais les valeurs X et Y sont toujours nulles.

Le premier dialogue est affiché avec sa méthode `Execute`. Ce dialogue comporte deux boutons, chacun déclenchant un gestionnaire spécifique sur l'événement Lors du déclenchement. De plus, la propriété `Tag` (Complément d'information) de chaque bouton contient le chiffre 2.

Étudions le gestionnaire `LancerDialogue`. Il utilise la valeur du `Tag`, d'une part pour reconstituer le nom de l'onglet du dialogue à utiliser, d'autre part pour servir d'index au tableau `Dlg`. Nous réutilisons la fonction utilitaire `Dialogue` et affichons le deuxième dialogue. Le gestionnaire `LancerDialogueCentre` reprend ce principe et ajoute l'appel à la routine `CenterDialog`. Nous lui transmettons en argument le nouveau dialogue et le précédent.

Le dialogue 2 est similaire au dialogue 1, avec deux boutons pour lancer le dialogue 3. Lorsqu'on appuie sur un des boutons, le mécanisme recommence. Si vous utilisez le même bouton pour lancer les dialogues successifs, le gestionnaire d'événement correspondant sera appelé indirectement de manière récursive. Si vous alternez les boutons, la récursion sera croisée entre les deux gestionnaires. Un codage plus lourd aurait pu éviter les récursions.

Nous arrêtons la démonstration au niveau 4. Maintenant, exécutez le programme plusieurs fois. Variez les appels, déplacez à la souris un dialogue avant d'appeler le suivant, étudiez l'importance de la position d'un dialogue dans l'EDI. Les figures 16-4 et 16-5 donnent deux exemples de résultats.

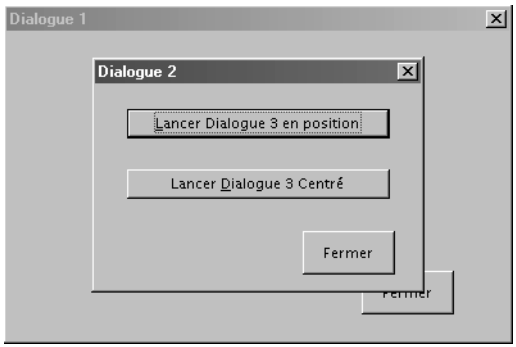


Figure 16–4
Deux dialogues centrés

Comme vous le verrez, un dialogue est toujours modal, c'est-à-dire qu'il bloque les dialogues de niveau inférieur. Vous devez fermer le dialogue pour manipuler la fenêtre du dialogue précédent.

> **BOGUE Position des dialogues**
>
> Vous constaterez parfois un comportement anormal de la position des dialogues de niveaux 3 et 4, dû à une faute logicielle dans OpenOffice.org. En fait, les dialogues emboîtés à un tel niveau ne sont pas vraiment testés.

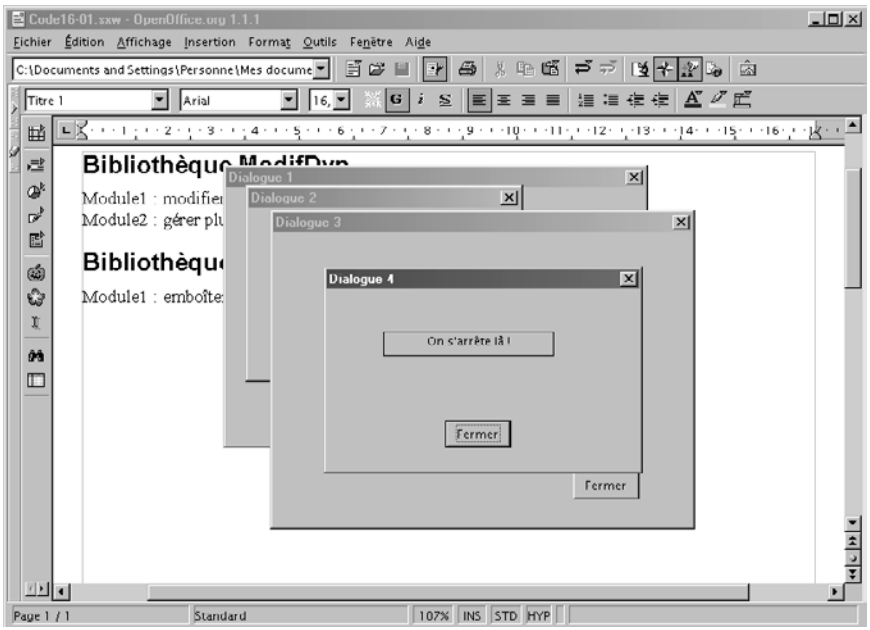


Figure 16–5
Quatre dialogues emboîtés

Les services de dialogues

Nous avons vu au chapitre 15 que le contrôle Sélection de fichiers (`FileControl`), placé dans un dialogue, donne à l'utilisateur le moyen de rechercher un fichier en explorant des répertoires. Nous avons aussi constaté qu'il est assez primitif.

Le service `FilePicker` offre des possibilités plus évoluées, sans même nécessiter une boîte de dialogue. En fait, pour l'utiliser dans un dialogue on devra l'exécuter sur un événement, par exemple suite au déclenchement d'un bouton.

Le service `FolderPicker` est analogue au précédent et sert à choisir un répertoire (appelé aussi dossier).

Les services de dialogues sont décrits en anglais dans l'IDL de la documentation SDK, à la page `com.sun.star.ui.dialogs`. Le terme `ui` est l'abrégé anglais pour interface utilisateur (*user interface*).

Sélectionner un fichier existant

Cet exemple vous montrera quelques possibilités appréciables.

```
rem Code16-02.sxw  bibli : ChoixFich Module1
Option Explicit

Sub OuvrirUnFichier()
Dim FP As Object, lesFichiers() As String
FP = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
With FP
  .DisplayDirectory = ConvertToURL("C:\Docs OpenOffice\")
  .appendFilter("Textes", "*.txt")
  .appendFilter("Docs OpenOffice", "*.sxw;*.sxc;*.sxd,*.sxi")
  .appendFilter("Docs MS-Office", "*.doc;*.xls;*.ppt")
  .CurrentFilter = "Docs OpenOffice"

  if .Execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK then
    lesFichiers() = .Files ' tableau de 1 élément String
    MsgBox("Filtre choisi = " & FP.CurrentFilter)
    MsgBox("Répertoire = " & .DisplayDirectory)
    MsgBox("Chemin complet = " & lesFichiers(0))
  end if
  .dispose
End With
End Sub
```

Nous demandons d'abord à l'API de nous fournir un objet `FilePicker FP` par l'intermédiaire de la fonction OOoBasic `CreateUnoService`. Il est utilisé comme un dialogue : la fonction `Execute` l'affiche et indique comment il a été fermé ; la méthode `dispose` libère la ressource. Ici, elle est peu utile car nous sommes en fin d'exécution. Avant d'afficher le dialogue, nous effectuons quelques initialisations (non obligatoires) :

- `DisplayDirectory` permet de choisir un répertoire initial. Il est nécessaire de lui fournir le chemin au format URL.
- La méthode `appendFilter` ajoute des filtres de visualisation, pour ne voir que les noms de fichiers correspondant au nom générique indiqué. On indique plusieurs filtres simultanés en les séparant par un point-virgule.
- La propriété `CurrentFilter` précise le filtre à utiliser initialement.

Lors de l'exécution (voir la figure 16-6), vous vérifierez l'efficacité de ces initialisations. Vous constaterez que le dialogue refuse un nom de fichier inexistant ; ceci est normal, car nous sommes dans un dialogue destiné à ouvrir un fichier, ce qui est confirmé par le titre du dialogue : `Ouvrir`. Si ce titre ne vous convient pas, il suffit d'en affecter un autre à la propriété `Title` du dialogue.

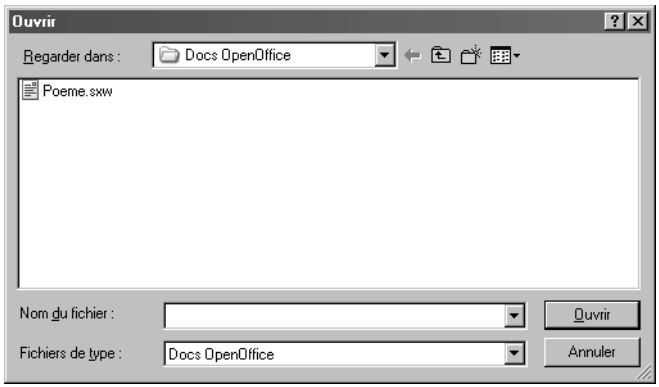


Figure 16–6
Dialogue de sélection de fichier

Vous ne pouvez choisir qu'un seul fichier et pourtant nous avons utilisé une variable tableau pour recueillir le nom du fichier fourni dans la propriété `Files`. La raison apparaîtra dans le prochain exemple. Ici, le tableau n'a qu'un seul élément, d'index zéro, contenant le chemin complet du fichier en syntaxe URL. Vous pouvez ensuite utiliser la fonction OOoBasic `ConvertFromURL` pour l'exprimer dans le format du système d'exploitation.

Les propriétés `CurrentFilter` et `DisplayDirectory` nous précisent ce qu'a choisi l'utilisateur. Remarquez le / final dans le contenu de `DisplayDirectory`.

Sélectionner plusieurs fichiers existants

Pour sélectionner plusieurs fichiers dans le même dialogue, il suffit de donner la valeur `True` à sa propriété `MultiSelectionMode`. Dans ce cas, la propriété `Files` renverra un tableau d'un ou plusieurs élément(s). Exécutez cet exemple et notez bien les résultats.

```
rem Code16-02.sxw  bibli : ChoixFich Module2
Option Explicit

Sub SelectionnerDesFichiers()
Dim FP As Object, lesFichiers() As String, x As Long
FP = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
With FP
  .DisplayDirectory = ConvertToURL("C:\Docs OpenOffice\")
  .appendFilter("Textes", "*.txt")
  .appendFilter("Docs OpenOffice", "*.sxw;*.sxc;*.sxd,*.sxi")
  .appendFilter("Docs MS-Office", "*.doc;*.xls;*.ppt")
```

```
  .CurrentFilter = "Docs OpenOffice"
  .MultiSelectionMode = True

  if .Execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK then
    lesFichiers() = .Files ' tableau de String
    MsgBox("Filtre choisi = " & FP.CurrentFilter)
    MsgBox("Répertoire = " & .DisplayDirectory)
    for x = LBound(lesFichiers) to UBound(lesFichiers)
      MsgBox("Fichier = " & lesFichiers(x))
    next
  end if
  .dispose
End With
End Sub
```

Remarque : les propriétés `Title`, `MultiSelectionMode`, `DefaultName`, sont en écriture seule.

Si vous avez choisi un seul fichier, le résultat est identique à l'exemple précédent. Si vous avez choisi plusieurs fichiers, le résultat est assez différent :

- La propriété `DisplayDirectory` nous donne un chemin sans le / final.
- Le premier élément de la variable `lesFichiers` est le chemin d'accès aux fichiers, sans le / final.
- Les éléments suivants ne comportent que le nom et l'extension de chaque fichier choisi.

Il appartient au programmeur de prévoir un codage en conséquence.

Enregistrer un fichier

Nous allons maintenant construire un dialogue `Enregistrer sous...`. Comme seul le dialogue nous intéresse, nous n'enregistrerons rien.

```
rem Code16-02.sxw  bibli : ChoixFich Module3
Option Explicit

Sub EnregistrerUnFichier()
Dim FP As Object, lesFichiers() As String
Dim FPtype(0) As Integer
FP = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
FPtype(0) = com.sun.star.ui.dialogs.TemplateDescription.FILESAVE_SIMPLE
With FP
  .initialize(FPtype())
  .DisplayDirectory = ConvertToURL("C:\Docs OpenOffice\")
  .DefaultName = "mon beau fichier.sxw"
  .appendFilter("Textes", "*.txt")
  .appendFilter("Docs OpenOffice", "*.sxw;*.sxc;*.sxd,*.sxi")
  .appendFilter("Docs MS-Office", "*.doc;*.xls;*.ppt")
  .CurrentFilter = "Docs OpenOffice"
```

```
  if .Execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK then
    lesFichiers() = .Files ' tableau de 1 élément String
    MsgBox("Filtre choisi = " & FP.CurrentFilter)
    MsgBox("Répertoire = " & .DisplayDirectory)
    MsgBox("Chemin complet = " & lesFichiers(0))
  end if
  .dispose
End With
End Sub
```

Nous utilisons la méthode `initialize` pour préciser quel modèle de dialogue nous allons utiliser. Plusieurs modèles sont possibles, comme que nous le verrons plus loin. Pour des raisons de conception de l'API, la méthode `initialize` n'accepte qu'un tableau de valeurs, mais dans le contexte des dialogues, seule une valeur sera utilisée.

La propriété `DefaultName` nous sert à proposer un nom de fichier. Une petite bogue fait que ce nom n'apparaît qu'après certaines manipulations de la boîte de dialogue.

Le titre du dialogue est, par défaut, `Enregistrer sous`, comme le montre la figure 16-7. Si vous choisissez un fichier existant, le dialogue vous demande une confirmation.

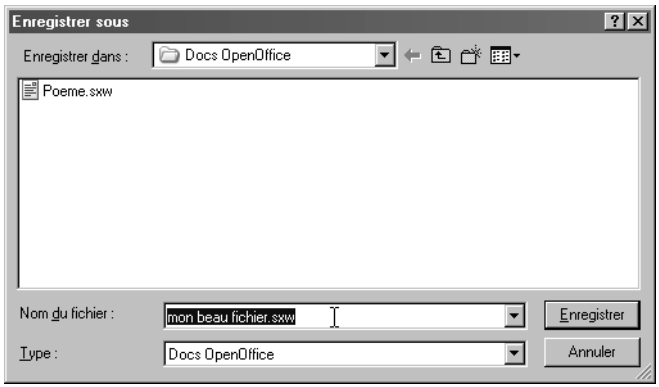


Figure 16–7
Dialogue d'enregistrement de fichier

Il est possible d'utiliser la sélection multiple, bien que cela ne cadre pas avec l'usage habituel.

Les différents dialogues de FilePicker

Les variantes de dialogues sont définies par des constantes nommées extrêmement longues, dont la racine est :

```
com.sun.star.ui.dialogs.TemplateDescription.
```

L'étude de ces variantes laisse une impression de développement logiciel encore en cours. Seulement trois d'entre elles sont utilisables facilement :

- `FILEOPEN_SIMPLE` qui correspond à la valeur par défaut lorsque la méthode `initialize` n'est pas employée,
- `FILESAVE_SIMPLE` qui a déjà été vue,
- `FILESAVE_AUTOEXTENSION` qui ajoute à la précédente une case à cocher Extension automatique du nom du fichier, qui utilise la première des extensions indiquées par le filtre.

Les autres dialogues nécessitent, pour être utilisables, des développements qui sortent du cadre de ce livre. Nous renvoyons les développeurs intéressés à la documentation API.

Choisir un répertoire

Le service `FolderPicker` est assez simple à utiliser.

```
rem Code16-02.sxw  bibli : ChoixRep Module1
Option Explicit


Sub ChoisirUnRepertoire()

Dim FP As Object

FP = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
With FP
  .DisplayDirectory = ConvertToURL("C:\Docs OpenOffice\")
  '.Description = "Cliquez sur un répertoire"
  '.Title = "Choisissez votre répertoire de sauvegarde"
  if .Execute = com.sun.star.ui.dialogs.ExecutableDialogResults.OK then
    MsgBox("Chemin = " & .Directory)
  end if
End With
End Sub
```

À son ouverture, le dialogue affichera le répertoire contenant celui qui est indiqué par sa propriété `DisplayDirectory`. En revanche, le répertoire choisi par l'utilisateur est obtenu dans la propriété `Directory`, en lecture seule.

Les textes affichés par défaut dans le titre du dialogue et dans la description peuvent être changés. Les instructions en commentaires dans l'exemple produisent le dialogue de la figure 16-8. Bizarrement, la méthode `dispose` n'existe pas dans l'objet `FolderPicker`.

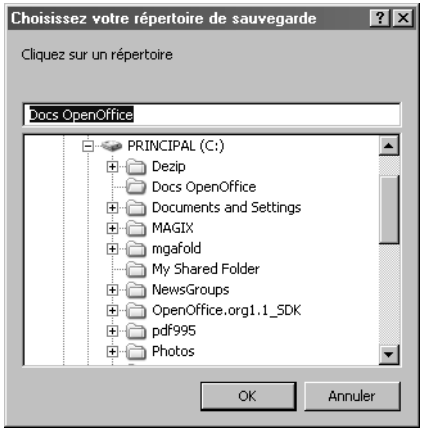


Figure 16–8
Dialogue de sélection de répertoire

Conclusion

Ce chapitre conclut la présentation des boîtes de dialogue. Les outils exposés permettent d'envisager de multiples utilisations pour créer des boîtes et assistants toujours plus simples et conviviaux pour l'utilisateur.

Toujours dans un esprit d'élargissement des possibilités offertes par les macros et l'API, le chapitre suivant va traiter de l'utilisation des sources de données.

17

Les sources de données

Un outil bureautique permet certes de rédiger des lettres et de construire des tableaux, mais il doit également pouvoir manipuler les données de l'utilisateur, données qu'on aura souvent intérêt à stocker dans une base de données. OpenOffice.org propose un module d'accès à ces données par l'intermédiaire de l'outil « source de données ».

Ce chapitre va nous permettre d'utiliser les outils mis à notre disposition pour intervenir sur ces sources de données et donc utiliser OpenOffice.org comme un moyen de visualisation et de modification de ces données. Certaines des manipulations présentées ci-après requièrent quelques connaissances de base du langage SQL et bien sûr des notions sur les bases de données.

La version 2.0 d'OpenOffice.org amène une grande évolution dans le concept de sources de données. Évolution et non révolution, comme nous le verrons. Les principales méthodes de manipulation du contenu des sources de données restent inchangées et les macros présentées s'exécutent aussi bien sous les versions 1.1.x que 2.0 d'OpenOffice.org. Les différences majeures sont dues à l'arrivée du nouveau moteur de base de données et des outils associés.

VOCABULAIRE

Une source de données est un concept OpenOffice.org. Elle permet d'accéder à une base de données (les tables), de mémoriser des requêtes prêtes à l'emploi et propose des liens vers des formulaires utilisant la source de données.

Afficher les outils intégrés

Avant de plonger dans la manipulation des sources de données, voici comment afficher directement les outils standards d'OpenOffice.org. Dans certains cas, ce niveau d'accessibilité peut suffire à l'utilisateur.

Le gestionnaire de sources de données

Le panneau de configuration des sources de données est affichable depuis Writer ou Calc. Il permet de configurer les sources de données, de les explorer, d'enregistrer des requêtes. Il est accessible par l'API depuis n'importe quel élément de la suite même, si l'option de menu n'est pas présentée. Ainsi, il est possible de l'afficher indépendamment du contexte quand l'utilisateur en a besoin.

Pour mettre ceci en évidence, ouvrez un nouveau document Draw ou Impress. Cet élément d'OpenOffice.org ne possède pas de menu permettant d'appeler la fenêtre. La macro suivante permet d'afficher le gestionnaire de sources de données (figure 17-1).

```
rem Code17-01.sxd  bibli : Standard Module1
rem Spécifique à la version 1.1.x
Option Explicit

Sub AfficherPanneauSdd()
dim Sdd as Object, Retour As Long
dim args(0) as new com.sun.star.beans.PropertyValue

Sdd=createUnoService( _
             "com.sun.star.sdb.DatasourceAdministrationDialog")

args(0).Name="Title"
args(0).value="Ceci est le titre de la fenêtre"

Sdd.initialize(args())
Retour=Sdd.execute()
select case Retour
  case com.sun.star.ui.dialogs.ExecutableDialogResults.OK
    ' Retour=1
    print "La fenêtre est fermée par le bouton OK"
  case com.sun.star.ui.dialogs.ExecutableDialogResults.CANCEL
    ' Retour=0
    print "La fenêtre est fermée par le bouton Annuler"
end select
End Sub
```

Dans un premier temps, nous créons l'objet `Sdd` en faisant appel au service `DataSourceAdministrationDialog`. Pour être utilisable, cet objet doit être initialisé en appelant sa méthode `initialize`. Le tableau de propriétés `args()` permet de définir les

propriétés du gestionnaire, comme le titre de la fenêtre. Enfin, la fenêtre est affichée en appelant la méthode `execute`. Notons que cet appel est bloquant : la macro ne se poursuivra qu'une fois la fenêtre fermée par l'utilisateur. La variable `Retour` représente l'état de la fermeture. Elle contient une des valeurs du groupe de constantes `ExecutableDialogResults`. Rappelons que dans le cas des constantes nommées, la casse est importante.

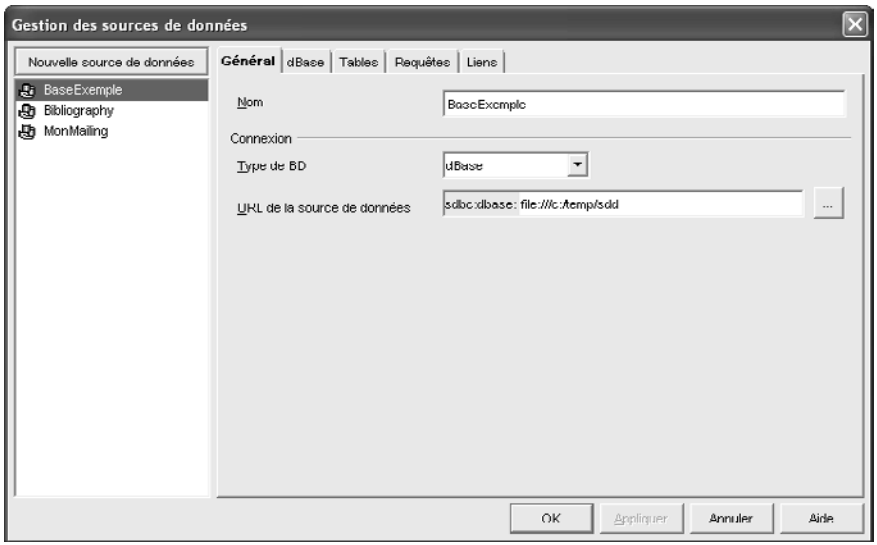


Figure 17–1
Gestionnaire de sources de données

L'assistant de création d'états

Bien que non documenté et peu automatisable, nous présentons dans le même esprit l'affichage de l'assistant d'état (figure 17-2). La macro suivante affiche la fenêtre du générateur d'état.

```
rem Code17-01.sxd  bibli : Standard Module2
rem Spécifique à la version 1.1.x
Option Explicit

Sub AfficherGenerateurEtat()
dim args(), leGenerateur as object

leGenerateur=createUnoService( _
              "com.sun.star.wizards.report.CallReportWizard")
leGenerateur.initialize(args())
End Sub
```

Nous créons tout d'abord un objet sur la base du service `CallReportWizard`. La fenêtre est affichée dès l'appel de la méthode `initialize`. Un document Writer pré-rempli va

alors s'ouvrir et l'assistant de génération d'état va s'afficher (figure 17-2). Là encore, l'appel est bloquant et nécessite l'intervention de l'utilisateur. Notons que le tableau `args()` n'a pas besoin d'être typé, car il ne contient aucune information.

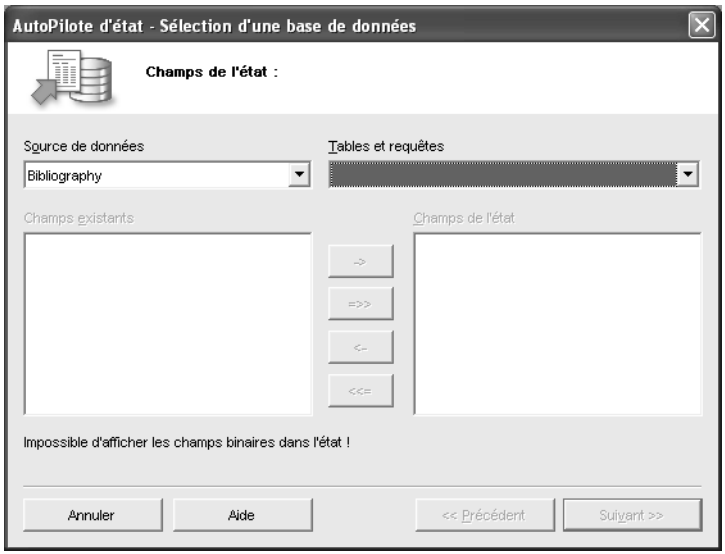


Figure 17–2
L'autopilote de création d'état

OpenOffice.org 2.0

La macro précédente ne donne un résultat satisfaisant que sous OOo 1.1.x. Pour la version 2.0, un écran obsolète sans aucune utilité s'ouvre.

Il reste néanmoins possible d'afficher le gestionnaire de sources de données en utilisant le *dispatcher*. Ce procédé est illustré par la macro suivante, que vous pouvez faire exécuter dans une version 2.0 de Draw par exemple.

```
rem Code17-01.odg  bibli : Standard Module1

Sub FermeSDD()
   call DispatchBrowser(false)
end sub

Sub OuvreSDD()
   call DispatchBrowser(true)
end sub


sub DispatchBrowser(ouvrir as Boolean)
dim document as object, dispatcher as object
dim args1(0) as new com.sun.star.beans.PropertyValue
```

```
document   = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
args1(0).Name = "ViewDataSourceBrowser"
args1(0).Value = ouvrir

dispatcher.executeDispatch(document, ".uno:ViewDataSourceBrowser", _
                           "", 0, args1())
end sub
```

Si l'option `args1(0) ViewDataSourceBrowser` est transmise avec l'argument `True`, la fenêtre sera affichée. Si l'argument est à `False`, elle sera fermée.

Draw n'ayant pas de menu explicite pour gérer les sources de données, il vous faudra définir une macro avec l'argument `False` pour fermer cette fenêtre.

Les sources de données

OpenOffice.org 1.1.x ne possède pas de moteur de bases de données à proprement parler. Il s'appuie sur des connecteurs à différents types de bases de données (ODBC, MySQL, Dbase...) pour relayer des requêtes aux moteurs externes.

OpenOffice.org 2.0 propose quant à lui un moteur de base de données embarqué HSQLDB. L'arrivée du module Base ne va avoir d'impact que pour la définition de la source de données. Les utilisations qui en découlent ne seront pas très différentes suivant la version d'OpenOffice.org.

Les objets utilisés au niveau des sources de données sont essentiellement regroupés dans les branches de l'API :

- `com.sun.star.sdb`
- `com.sun.star.sdbc`
- `com.sun.star.sdbcx`

> Le chapitre 12 du « Developer's Guide » du SDK est consacré à la gestion des sources et bases de données.

Afin de manipuler directement les sources de données, nous abandonnons les deux approches présentées précédemment et nous nous intéressons désormais réellement aux objets de sources de données que nous procure l'API.

Lister les sources de données

Tout accès aux sources de données est basé sur un contexte de base de données disponible par le service `DatabaseContext`. On crée donc un objet `MonDbContext` basé sur ce service, à

l'aide de l'instruction `CreateUnoService`. L'objet `MonDbContext` est l'objet principal à partir duquel nous allons utiliser les sources de données. Il est équivalent à la fenêtre de l'interface graphique à partir de laquelle toutes les sources de données sont accessibles. Cet objet nous fournit le tableau `ElementNames`, qui contient le nom des sources de données disponibles.

```
rem Code17-02.sxw  bibli : Sources Module1
Option Explicit

Sub ListerNomDesSourcesDeDonnees()
dim MonDBContext As Object
dim LeTableau() As String, i as long

MonDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
LeTableau=MonDbContext.ElementNames
For i = lbound(LeTableau) to ubound(LeTableau)
  print LeTableau(i)
next i
End Sub
```

Pour accéder à chacun des objets de la collection contenue dans `MonDBContext`, nous utiliserons une deuxième approche.

```
rem Code17-02.sxw  bibli : Sources Module2
Option Explicit

Sub ListeSourcesDeDonnees()
dim MonDBContext As Object
dim MonEnum As Object, UneSource As Object

MonDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
MonEnum = MonDbContext.createEnumeration()
while MonEnum.hasMoreElements
  UneSource = MonEnum.nextElement
  print UneSource.name
wend
End Sub
```

En utilisant l'énumération `MonEnum` retournée par la méthode `createEnumeration`, nous constituons une liste des sources de données accessibles. `MonEnum` peut donc être vu comme un tableau constitué des sources de données présentes au niveau d'OpenOffice.org. Cependant, il s'agit d'une séquence, et non d'un tableau, et son utilisation est un peu différente.

Toute énumération retournée par l'API possède en effet des méthodes de navigation permettant de savoir si en parcourant cette liste, il reste des éléments auxquels accéder (le booléen `hasMoreElements`) et d'accéder à l'objet suivant (par l'utilisation de `nextElement`).

Ainsi, la boucle `while` nous permet de parcourir cette énumération et d'obtenir à chaque boucle une source de données différente. Pour connaître celle que nous manipulons, nous affichons sa propriété `name`. Nous reviendrons plus loin sur les propriétés des sources de données.

> **À RETENIR Énumération**
>
> De nombreux objets collections capables d'énumérer leur contenu existent dans l'API. La même méthode que celle décrite ici est alors employée.

Non seulement l'objet `MonDbContext` sait énumérer les sources de données, mais il connaît aussi leur nom. Ainsi, il possède également la méthode `hasByName` permettant d'indiquer si une source de donnée est présente. Cette possibilité s'avèrera très intéressante quand nous créerons dynamiquement une source de données afin d'éviter une duplication du nom (ce qui est interdit par l'API). L'objet `MonDbContext` est enfin capable de trouver une source de données à partir de son nom (méthode `getByName`).

Propriétés d'une source de données

Avant d'aller plus avant, voici quelques propriétés accessibles au niveau de la source de données. Nous ne donnons ici que les plus courantes, en utilisant la source de données `Bibliography` présente en standard dans OpenOffice.org.

```
rem Code17-02.sxw  bibli : Sources Module3
Option Explicit

Sub ProprieteSource()
dim NomSource As String, NomPropriete As String, cr As String
dim monDBContext As Object, maSource As Object
dim Reponse As Boolean

cr = chr(13) ' retour à la ligne
NomSource="Bibliography"
monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")

Reponse = monDbContext.hasByName(NomSource)
if Reponse then
  maSource = monDbContext.getByName(NomSource)
  MsgBox("Nom = " & maSource.Name & cr & _
    "URL = " & maSource.URL & cr & _
    "Mot de passe requis ? " & maSource.IsPasswordRequired & _
    cr & "En lecture seule ? " & maSource.IsReadOnly & cr & _
    "Utilisateur = " & maSource.User & cr & _
    "Mot de passe = " & maSource.Password)
else
  print "La source '" & NomSource & "' n'existe pas"
endif
End Sub
```

Après avoir créé notre contexte de source de données, nous vérifions que la source de données `Bibliography` existe bien, grâce à la fonction `hasByName` qui renvoie `True` dans ce cas.

Puis, en utilisant la fonction `getByName`, nous définissons la variable `maSource` attachée à la source de données `Bibliography`. Nous appelons ensuite les différentes propriétés.

Une liste plus complète est reproduite dans le tableau 17-1.

Tableau 17–1 Propriétés de la source de données

Propriété	Type	Signification
`Name`	`String`	Nom de la source de données.
`URL`	`String`	Chemin de la source de données. La structure de cette URL contient les protocoles utilisés, elle est exposée plus loin.
`IsPasswordRequired`	`Boolean`	`True` si un mot de passe est requis.
`IsReadOnly`	`Boolean`	`True` si la source de données est déclarée comme étant en lecture seule.
`User`	`String`	Nom de l'utilisateur.
`Password`	`String`	Mot de passe.
`LoginTimeOut`	`Long`	Nombre de secondes avant la déclaration d'un échec de connexion.

Il existe d'autres propriétés, comme `TableFilterType`, permettant de filtrer les tables (`TABLES`), vues (`VIEW`) et tables systèmes (`SYSTEM TABLE`) par exemple. Nous laissons le lecteur regarder leur description dans la définition du service `com.sun.star.sdb.DataSource`.

Créer et supprimer une source de données

Le zip téléchargeable contient dans le répertoire `MacrosLivre/Ch17/sdd` une base de données au format dBase, qui reprend les tables classiques pour la gestion d'un commerce : client, produits, ventes. Nous vous conseillons de la copier sur votre disque dur afin de pouvoir travailler dessus et dans un premier temps l'installer.

> **REMARQUE Limitations de dBase**
>
> L'utilisation du format dBase fournit une base commune de tests pour nos exemples, mais il est insuffisant pour bien des usages. Vous ne pourrez profiter pleinement de la puissance d'OpenOffice.org 1.1.x qu'en utilisant une vraie base de données relationnelle, type MySQL, où les jointures sont autorisées et le langage SQL pleinement reconnu.
>
> Pour OpenOffice.org 2.0, l'utilisateur pourra employer la nouvelle base de données embarquée HSQLDB.
>
> Nous resterons avec le format dBase afin de pouvoir exécuter les exemples également dans la version 1.1.x.
>
> Les principes exposés dans ce chapitre sont en effet indépendants du moteur de base de données utilisé.

Créer une source de données revient avant tout à indiquer à OpenOffice.org le chemin d'accès aux données ainsi que le protocole utilisé, sous la forme d'une URL. Open-

Office.org propose un grand nombre de connecteurs aux sources de données qui vont influer sur la syntaxe de cette URL. L'URL va se découper ainsi : `<protocole>:<sous-protocole>:<chemin>`

Les protocoles reconnus sont :

- `sdbc` – celui que nous utiliserons par la suite
- `JDBC`

Les sous-protocoles définissent le type de la base de données :

- `MySQL`
- `Adabas`
- `ODBC`
- `dBase` – celui que nous utiliserons par la suite
- `ADO`
- `texte`
- `calc` (un tableur Calc)
- `Carnet d'adresses`
- `embedded:hsqldb` pour la base embarquée d'OpenOffice.org 2.0

Avec OpenOffice.org 1.1.x, les différentes combinaisons `<protocole>:<sous-protocole>` peuvent être explorées dans le gestionnaire de sources de données. Ce n'est plus possible en version 2.0. Les URL complètes avec le protocole `sdbc` sont les suivantes :

- document Calc : `sdbc:calc:URLfichierCalc`
- fichier texte plat : `sdbc:flat:URLfichierTexteOuRepertoire`
- dBase : `sdbc:dbase:URLfichierDBaseOuRepertoire`
- carnet d'adresses Mozilla : `sdbc:address:mozilla`
- carnet d'adresses LDAP : `sdbc:address:ldap`
- carnet d'adresses Outlook Express (MS-Windows) : `sdbc:address:outlookexp`
- carnet d'adresses Outlook (MS-Windows) : `sdbc:address:outlook`
- base embarquée dans OOo 2.0 : `sdbc:embedded:hsqldb`

Ainsi, pour notre exemple, si vous copiez le répertoire `sdd` vers `<MonRepertoire>/sdd`, l'URL de la source de données sera : `sdbc:dbase:file:///<MonRepertoire>/sdd`

OpenOffice.org 1.1.x

Nous allons donc à présent créer la source de données à partir de ce répertoire sous le nom `BaseExemple`. Veillez à mettre la variable `Chemin` en accord avec l'emplacement physique du répertoire `sdd`.

```
rem Code17-02.sxw  bibli : Sources Module4
Option Explicit

Sub CreerSource()
dim monDBContext As Object, uneInstance As Object
Dim Nomsource As String, Chemin As String

NomSource="BaseExemple"
monDBContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
if not monDBContext.hasByName(NomSource) then
  Chemin = convertToURL("c:\temp\sdd")
  uneInstance = monDBContext.createInstance()
  monDBContext.registerObject(NomSource, uneInstance)
  uneInstance.URL =  "sdbc:dbase:" & chemin
else
  MsgBox("Ce nom de source existe déjà." & chr(13) & _
         "Impossible de créer la nouvelle source.", 16)
endif
End Sub
```

Après avoir créé notre contexte de source de données, nous vérifions que la source de donnée n'existe pas déjà. Il est en effet impossible de créer deux sources de données sous le même nom. C'est donc l'objet du test avec la méthode `hasByName`.

Si cette source de donnée n'existe pas, alors nous définissons le chemin où est stocké notre base. Il est à noter que certains sous-protocoles comme `ODBC` ne requièrent pas un chemin mais un nom connu de celui-ci.

Nous avons vu que `monDBContext` était un conteneur regroupant toutes les sources de données. La procédure pour y insérer une nouvelle source de données est de créer une instance vierge par la méthode `createInstance` et de l'enregistrer sous un certain nom avec la méthode `registerObject`. Enfin nous spécifions l'URL de cette nouvelle source de données. Notez que c'est dans cette propriété que nous définissons le protocole qu'elle va utiliser (`"sdbc:dbase:"`).

Pour supprimer une source de données, il suffit d'appeler la méthode `revokeObject` sur le nom de la source de données (si vous utilisez cette macro, pensez à recréer la source de données pour les chapitres suivants). Attention, aucune confirmation n'est demandée lors de l'utilisation de la méthode `revokeObject`. Elle est donc à utiliser avec prudence.

```
rem Code17-02.sxw  bibli : Sources Module5
Option Explicit

Sub SupprimerSource()
dim monDBContext As Object, Nomsource As String
NomSource = "BaseExemple"
monDBContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
if monDBContext.hasByName(NomSource) then
  monDBContext.revokeObject(NomSource)
```

```
else
  MsgBox("Cette source de données n'existe pas", 16)
endif
End Sub
```

OpenOffice.org 2.0

Avec l'apparition du module Base, la méthodologie de création des sources de données a changé : un fichier `.odb`, format base de données d'OpenOffice.org, est systématiquement créé. Ce fichier contient la base native HSQLDB ou bien propose une indirection vers une autre base, dBase par exemple.

La source de données créée est alors listée dans le menu Outils>Options>OpenOffice.org Base> Sources de données.

```
rem Code17-01.odg  bibli : SourcesDeDonnees Module1
Option Explicit

Sub CreerSourceV2()
dim monDBContext As Object, uneInstance As Object
Dim Nomsource As String, Chemin As String
Dim monDocBase As Object

NomSource="BaseExemple"
monDBContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
if not monDBContext.hasByName(NomSource) then
  Chemin = convertToURL("c:\temp\sdd")
  uneInstance = monDBContext.createInstance()
  monDocBase = uneInstance.DatabaseDocument
  monDocBase.storeAsURL(convertToURL("c:\temp\sdd\maPetiteBase.odb"), Array())
  monDBContext.registerObject(NomSource, uneInstance)
  ' c'est cette ligne qui va définir le type de base de données sous-jacent
  uneInstance.URL =  "sdbc:dbase:" & chemin
  ' remplir éventuellement autres propriétés : IsPasswordRequired, User, etc
  monDocBase.store ' mémoriser les caractéristiques
else
  MsgBox("Ce nom de source existe déjà." & chr(13) & _
         "Impossible de créer la nouvelle source.", 16)
endif
End Sub
```

Après avoir créé notre contexte de source de données, nous vérifions que cette dernière n'existe pas déjà. Il est en effet impossible de créer deux sources de données sous le même nom. C'est donc l'objet du test avec la méthode `hasByName`. Si ce nom n'est pas utilisé, alors nous pouvons créer notre source de données.

Comme pour OOo 1.1.x, nous commençons par créer un exemplaire d'objet de contexte de base de données monDBContext.

Sous OOo 2, cet exemplaire expose un objet `DatabaseDocument` avec lequel nous allons définir le fichier OOoBase conteneur de notre source de données. Ce fichier est enregistré grâce à sa méthode `storeAsUrl`. Nous proposons de l'enregistrer au même niveau que votre base dBase.

La source de données est enregistrée, déclarée au niveau d'OpenOffice.org en utilisant la méthode `registerObject` du contexte `monDBContext`.

Il est alors possible de modifier ses propriétés et notamment son type défini par la propriété `URL`. Dans le cas d'une base dBase, cette URL est constituée du préfixe `sdbc:dbase:` suivi de l'URL du répertoire contenant les fichiers. Pour une base native, l'URL à utiliser est : `sdbc:embedded:hsqldb`

Afin de sauvegarder nos modifications sur la structure de notre source de données, il est nécessaire de les stocker au niveau du fichier `.odb` conteneur que nous venons de créer. Nous appelons donc la méthode `store` de l'objet `DatabaseDocument` de notre contexte.

À partir de ce point, notre source de données est créée, que ce soit pour une source externe comme dBase ou MySQL, ou bien pour la base de données native d'OpenOffice.org 2.0.

La méthodologie de suppression de la source de données est identique à celle de la version 1.1.x. Il suffit d'appeler la méthode `revokeObject` de la source de données obtenue par l'intermédiaire du contexte de source de données. Le même avertissement s'applique : aucune demande de confirmation n'est affichée.

Enfin, OOo 2 Base s'appuyant sur un fichier éventuellement intermédiaire, il pourra être nécessaire de le supprimer pour nettoyer toute trace de la définition de la source de données, en utilisant l'instruction `Kill` de Basic par exemple.

Se connecter à une source de données

Maintenant que nous disposons d'une source de données spécifique déclarée dans OpenOffice.org, nous allons voir comment l'utiliser avec les ressources de l'API.

La première étape est de demander explicitement à OpenOffice.org de se connecter à la source en question, c'est-à-dire obtenir un objet permettant de l'utiliser. L'exemple `ConnecterDeconnecter` présenté ici utilise deux macros que nous réutiliserons tout au long de ce chapitre.

```
rem Code17-02.sxw  bibli : BaseDonnees Module1
Option Explicit

Sub ConnecterDeconnecter()
ConnecterSource
print maConnexion.parent.name
DeconnecterSource
End Sub
```

Les routines `ConnecterSource`, `DeconnecterSource`, ainsi que la variable publique `maConnexion` sont définies dans le module `Standard` afin d'être utilisables par les macros situées dans d'autres bibliothèques du document exemple.

```
rem Code17-02.sxw  bibli : Standard Module1
Option Explicit

Public maConnexion As Object

Sub ConnecterSource()
Dim NomSource As String, login As String, password As String
Dim maSource As Object, monDbContext As Object

'Création du contexte
NomSource = "BaseExemple"
monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource=monDbContext.getByName(NomSource)
'Les paramètres de connexion
login = ""
password = ""
maConnexion = maSource.getConnection(login, password)
if IsNull(maConnexion) then
  MsgBox("Connexion impossible", 16)
  Stop
end if
End Sub

Sub DeconnecterSource()
maConnexion.close
maConnexion.dispose
End Sub
```

Nous commençons donc par créer le contexte sur notre base. Nous nous servirons ici de la base dBase que nous avons enregistrée sous le nom de `BaseExemple`.

Généralement, les bases de données requièrent une identification à qui veut s'y connecter : nom d'utilisateur et mot de passe. Nous définissons donc deux variables permettant de passer ces informations. Dans notre cas, elles sont vides puisque la base utilisée n'en requiert pas. Il est à noter qu'il n'est pas non plus nécessaire de les spécifier si la source de données utilise le protocole `ODBC` et que ces informations y sont stockées. Dans le cas général, si le mot de passe est incorrect nous obtiendrons un objet `Null`. Pour simplifier, dans ce cas nous affichons un message et arrêtons la macro.

Afin d'obtenir une connexion, nous utilisons la méthode `getConnection` du contexte, en lui passant les arguments d'identification. Nous avons ainsi établi une connexion à notre source de donnée. Nous affichons son nom. Pour fermer la connexion, nous appelons la routine `DeconnecterSource`. Elle appelle la méthode `close` de l'objet `maConnexion`, puis détruit l'objet en appelant la méthode `dispose`.

Demande de mot de passe à l'utilisateur

Si la source de données est protégée par un mot de passe, vous pouvez le demander à l'utilisateur avec une boîte de dialogue standardisée. Voici une deuxième version de la routine `ConnecterSource` :

```
rem Code17-02.sxw  bibli : Standard Module2
Option Explicit

Sub ConnecterSource_MotdePasse()
Dim NomSource As String, demandePasse As Object
Dim maSource As Object, monDbContext As Object

'Création du contexte
NomSource = "BaseExemple"
monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource=monDbContext.getByName(NomSource)
demandePasse = CreateUnoService("com.sun.star.sdb.InteractionHandler")

' nous forçons l'usage du mot de passe pour cet exemple
maSource.IsPasswordRequired = true

maConnexion = maSource.connectWithCompletion(demandePasse)
if IsNull(maConnexion) then
  MsgBox("Connexion impossible", 16)
  Stop
end if
End Sub
```

Le service `InteractionHandler` nous fournit un objet permettant le dialogue avec l'utilisateur. Pour les besoins de l'exemple, nous rendons artificiellement obligatoire le mot de passe, car notre source ne le nécessite pas d'origine. La connexion est obtenue avec la fonction `connectWithCompletion` de l'objet source. La figure 17-3 montre le dialogue affiché.

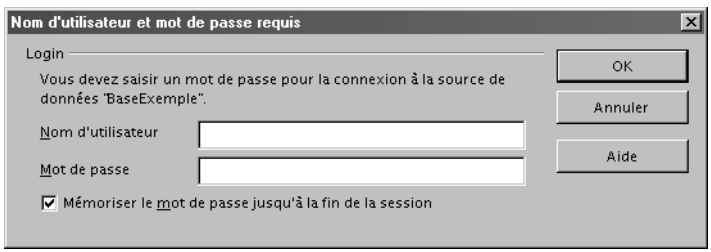


Figure 17–3
La boîte de dialogue d'authentification

Comme notre source n'a pas de mot de passe, il suffit de cliquer sur `OK` pour se connecter. Si l'utilisateur clique sur `Annuler`, la fonction `connectWithCompletion` renvoie un objet `Null`.

Propriétés d'une connexion

L'objet `maConnexion` va désormais être notre objet de travail principal ; c'est pourquoi nous l'avons déclaré `Public`. Il expose plusieurs propriétés et méthodes que nous allons explorer dans les sections suivantes. Notons qu'il propose un objet `parent` qui tout naturellement nous retourne le `databaseContext` dont il est issu, ce que nous vérifions en appelant la propriété `name` de l'objet `parent`.

Enfin, certaines propriétés relatives au pilote utilisé et la syntaxe SQL reconnue, entre autres, sont accessibles par la propriété `MetaData`. Ces informations étant assez techniques (comme le nom du pilote, sa version...), nous fournissons une macro pour en lister quelques-unes, mais nous ne rentrerons pas dans les détails.

```
rem Code17-02.sxw  bibli : BaseDonnees Module2
Option Explicit

Sub afficherMetaData()
Dim LesMetaDatas As Object

ConnecterSource
LesMetaDatas = maConnexion.getMetaData
with LesMetaDatas
  print "URL : " & .URL
  print "UserName : " & .UserName
  print "DatabaseProductName : " & .DatabaseProductName
  print "DatabaseProductVersion : " & .DatabaseProductVersion
  print "DriverName : " & .DriverName
  print "DriverVersion : " & .DriverVersion
  print "DriverMajorVersion : " & .DriverMajorVersion
  print "DriverMinorVersion : " & .DriverMinorVersion
  print "IdentifierQuoteString : " & .IdentifierQuoteString
  print "SQLKeywords : " & .SQLKeywords
  print "NumericFunctions : " & .NumericFunctions
  print "StringFunctions : " & .StringFunctions
  print "SystemFunctions  : " & .SystemFunctions
  print "TimeDateFunctions : " & .TimeDateFunctions
end with

DeconnecterSource
End Sub
```

> Vous trouverez plus d'informations sur le contenu de `MetaData` dans la documentation API, à l'interface `com.sun.star.sdbc.XDatabaseMetaData`.

Bien entendu ce type d'information trouvera toute sa pertinence lors de l'utilisation d'une vraie base de données relationnelle (ce qui n'est pas le cas avec `baseExemple`), les informations accessibles étant extrêmement nombreuses.

Nous en donnons un très bref aperçu dans le tableau 17-2.

Tableau 17–2 Quelques propriétés de MetaData

Propriété	Type	Signification
URL	String	Répertoire et nom de fichier de la base de données.
UserName	String	Identifiant de l'utilisateur connecté.
DatabaseProductName	String	Type du moteur de la base de données.
DatabaseProductVersion	String	Version du moteur de la base.
DriverName	String	Nom du pilote utilisé.
DriverVersion	String	Version du pilote.
SQLKeywords	String	Liste des mots-clés SQL reconnus et n'appartenant pas à la norme SQL92.
NumericFunctions	String	Liste des fonctions mathématiques reconnues dans la syntaxe SQL.
StringFunctions	String	Liste des fonctions de manipulation de chaînes de caractères reconnues dans la syntaxe SQL.
TimeDateFunctions	String	Liste des fonctions date/heure reconnues dans la syntaxe SQL.
SupportsGroupBy	Boolean	True si la clause GroupBy est reconnue.
SupportsTransaction	Boolean	True si les transactions sont reconnues.

La connexion sur une source de données donne accès à plusieurs collections permettant de visualiser ou manipuler sa structure. Nous nous intéresserons maintenant à la collection `Tables` et à la collection `Queries` qui, elle, donne accès aux requêtes pré-enregistrées.

Les tables

La collection `Tables`, comme son nom l'indique, nous donne accès à des objets `Table` (`com.sun.star.sdbcx.Table`) décrivant la structure de la base. Dans une base de données, chaque table a un nom et contient des champs possédant chacun un nom et un type. Donc, un objet `Table` contient lui-même une collection `Columns`, les champs. Cette collection contient quant à elle, les objets `Column` (`com.sun.star.sdbcx.Column`) donnant accès aux propriétés des champs.

L'exemple suivant va nous permettre d'illustrer simplement les imbrications énoncées. Il utilise deux routines que nous détaillerons successivement.

```
rem Code17-02.sxw  bibli : BaseDonnees Module3
Option Explicit

Sub InfoTables()
Dim LesTables As Object, LaTable As Object
Dim i As Long, NombreTables As Long
```

```
ConnecterSource
LesTables = maConnexion.tables 'Collection des tables de la base
NombreTables = LesTables.count
print "Nombre de Tables : " & NombreTables

for i=0 to NombreTables -1 'Boucle sur les tables
   LaTable = maConnexion.tables(i)
   print "Examen de la table : " & Latable.name
   AnalyserPrivileges(laTable.Privileges)
   AfficherInfoChamps(LaTable)
next i
DeconnecterSource
End Sub
```

Nous récupérons la collection des tables dans la variable `LesTables`. Afin de parcourir cette collection, nous aurions pu utiliser la méthode `createEnumeration` comme vu précédemment. Cependant, la collection donnant accès à la propriété `Count`, il nous suffit de l'utiliser dans une simple boucle `FOR` pour analyser chacune des tables. Nous constatons que la collection `LesTables` n'est rien d'autre qu'un simple tableau dont chacun des éléments représente une table spécifique dénommée ici `LaTable`.

Comme toute collection nommée, `LesTables` possède les méthodes `getElementNames` donnant la liste des noms des tables, `hasByName` indiquant si une table est présente et permettant d'accéder directement à une table par son nom (`getByName`) ou par son index (`getByIndex`).

```
dim ListeNoms() As String
dim reponse As Boolean
dim uneTable As Object

'Liste des noms des tables
ListeNoms = LesTables.getElementNames

'la table achats existe-t-elle
reponse = LesTables.hasByName("achats")

if reponse then
   'Accéder à une table
   uneTable = LesTables.getByName("achats")
endif

'Accéder à la dernière table de la collection
uneTable=lesTables.getByIndex(LesTables.count-1)
'autre syntaxe valide
uneTable=lesTables(LesTables.count-1)
```

Avant de manipuler une table, il est nécessaire de connaître les droits de l'utilisateur avec lequel nous sommes connectés. C'est la propriété `Privilege` qui va nous renseigner.

```
rem Code17-02.sxw  bibli : BaseDonnees Module3

Sub AnalyserPrivileges(lesPriv As Long)
dim chaine As String

chaine = "Privileges = " & hex(lesPriv)
if (lesPriv and com.sun.star.sdbcx.Privilege.SELECT) then
  chaine = chaine & chr(13) & "SELECT"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.INSERT) then
  chaine = chaine & chr(13) & "INSERT"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.UPDATE) then
  chaine = chaine & chr(13) & "UPDATE"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.DELETE) then
  chaine = chaine & chr(13) & "DELETE"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.READ) then
  chaine = chaine & chr(13) & "READ"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.CREATE) then
  chaine = chaine & chr(13) & "CREATE"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.ALTER) then
  chaine = chaine & chr(13) & "ALTER"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.REFERENCE) then
  chaine = chaine & chr(13) & "REFERENCE"
endif
if (lesPriv and com.sun.star.sdbcx.Privilege.DROP) then
  chaine = chaine & chr(13) & "DROP"
endif
```

La propriété `Privileges` de l'objet `LaTable` retourne un entier `Long` qui est l'addition de constantes représentant les actions du groupe `com.sun.star.sdbcx.Privilege`. Comme ces constantes sont des puissances de 2, à l'aide d'un ET logique (`AND`) nous pouvons savoir si telle ou telle action est autorisée. Nous affichons en hexadécimal la valeur de la propriété `Privileges`, en utilisant la fonction `hex` de Basic.

Les Champs

Dans la boucle de la macro `InfoTables` parcourant les tables de la collection, nous appelons la fonction `AfficherInfoChamps` avec l'objet `LaTable` en cours, afin de parcourir les champs de celle-ci.

```
rem Code17-02.sxw  bibli : BaseDonnees Module3

Sub AfficherInfoChamps(uneTable)
dim lesColonnes As Object, laColonne As Object
dim i As Long, nombreColonnes As Long, chaine As String

chaine = "Champs de la table"
lesColonnes = uneTable.columns ' Collection des champs
'Nombre de champs dans la table
nombreColonnes = lesColonnes.count

for i = 0 to nombreColonnes -1  'Boucle sur les champs
  laColonne = lesColonnes(i)
  chaine = chaine & chr(13) & laColonne.name & " - >   "
  chaine = chaine & "Type = " & laColonne.TypeName & "    "
  chaine = chaine & "Auto = " &  laColonne.isAutoIncrement

next i
msgbox chaine
End Sub
```

Cette macro est basée sur le même principe que la précédente : on récupère la collection `lesColonnes`, on détermine le nombre d'éléments du tableau par `Count` et on boucle sur chacune des colonnes pour obtenir un objet `laColonne`.

Pour chacun des champs, nous concaténons dans la variable `chaine` son nom `name`, son type `TypeName` et l'indication si le champ est auto-incrémenté `isAutoIncrement`. La variable `chaine` est ensuite affichée à la sortie de la routine, quand tous les champs ont été parcourus. D'autres propriétés peuvent être manipulées (voir tableau 17-3). Elles sont décrites, entre autres, dans `com.sun.star.sdbcx.Column`.

Tableau 17–3 Propriétés de Column

Propriété	Type	Signification
`Name`	`String`	Nom du champ
`Type`	`Long`	Type SQL du champ.
`TypeName`	`String`	Nom du type SQL.
`Precision`	`Long`	Nombre de décimales.
`IsNullable`	`Long`	Précise si le champ accepte les valeurs `NULL`.
`IsAutoIncrement`	`Boolean`	Le champ est auto-incrémenté (numérotation automatique).
`Description`	`String`	Description du champ.
`DefaultValue`	`String`	Valeur par défaut - Toujours sous forme de chaîne de caractères.

La propriété `IsNullable` n'est pas un `Boolean`, mais contient une constante nommée. Les trois valeurs possibles correspondent respectivement à Oui, Non, indéterminé :

```
com.sun.star.sdbc.ColumnValue.NULLABLE
com.sun.star.sdbc.ColumnValue.NO_NULLS
com.sun.star.sdbc.ColumnValue.NULLABLE_UNKNOWN
```

La propriété `Type` est une constante nommée de la forme :

```
com.sun.star.sdbc.DataType.VARCHAR
```

Les valeurs possibles sont listées dans le tableau 17-4. Leur définition est basée sur JDBC 2.0.

Tableau 17–4 Constantes de type de champ

Constante	Constante	Constante	Constante
BIT	DOUBLE	TIME	OBJECT
TINYINT	NUMERIC	TIMESTAMP	DISTINCT
SMALLINT	DECIMAL	BINARY	STRUCT
INTEGER	CHAR	VARBINARY	ARRAY
BIGINT	VARCHAR	LONGVARBINARY	BLOB
FLOAT	LONGVARCHAR	SQLNULL	CLOB
REAL	DATE	OTHER	REF

Notons également que la collection `lesColonnes` permet les mêmes méthodes d'accès par nom et par index que celles illustrées pour les tables : `hasByName`, `getByName`, etc.

Créer une table

Si la connexion nous y autorise, nous pouvons ajouter une table par macro. Cette table, pour être définie, doit au moins contenir un champ. Nous en ajouterons deux de types différents.

```
rem Code17-02.sxw  bibli : BaseDonnees Module4
Option Explicit

Sub CreerTable()
dim  descrTable As Object, lesTables As Object, descrCol As Object

ConnecterSource
lesTables = maConnexion.tables 'Collection des tables
print lesTables.Count 'Nombre de tables avant insertion

'Creation d'un descripteur de table
descrTable = lesTables.createDataDescriptor
descrTable.Name = "UneNouvelleTable"
```

```
'Ajout d'un champ au descripteur de table
descrCol = descrTable.columns.createDataDescriptor
descrCol.Name = "Champ1"
descrCol.Type = com.sun.star.sdbc.DataType.VARCHAR
descrCol.Precision = 100 ' nombre maximal de caractères
descrTable.Columns.appendByDescriptor(descrCol)

'Ajout d'un autre champ au descripteur de table
descrCol = descrTable.columns.createDataDescriptor
descrCol.Name = "Champ2"
descrCol.Type = com.sun.star.sdbc.DataType.DOUBLE
descrTable.Columns.appendByDescriptor(descrCol)

'Création de la nouvelle table
lesTables.appendByDescriptor(descrTable)

print lesTables.Count 'Nombre de tables après insertion
DeconnecterSource
End Sub
```

À partir de la collection `lesTables`, nous créons un descripteur de tables, `descrTable`, par la méthode `createDataDescriptor`. Il représente la table que nous allons créer et que nous ajouterons à la collection. La propriété `Name` renseigne le nom de la nouvelle table. Celui-ci doit bien sûr être unique. Notons que les descripteurs exposent les mêmes propriétés que les objets réels qu'ils représentent (autant les tables que les colonnes).

Une table doit contenir au moins un champ. Nous accédons donc à la collection `Columns` de ce descripteur pour créer, par sa méthode `createDataDescriptor`, un descripteur de champ `descrCol`. Nous en remplissons les propriétés `Name` (nom unique obligatoire) et `Type`. Les valeurs de `Type` sont des constantes nommées, décrites plus haut. Pour un champ du type `VARCHAR`, certaines bases de données comme HSQLDB exigent une taille maximale, définie par la propriété `Precision`.

Nous définissons ainsi suffisamment le champ pour qu'il soit créé. Nous l'ajoutons à la collection `columns` du descripteur de table en utilisant la méthode `appendByDescriptor` de celle-ci.

Nos deux champs créés, nous pouvons créer la table elle-même en ajoutant le descripteur de table à la collection des tables par la méthode `appendByDescriptor` de la collection `lesTables`.

Pour effacer une table, utilisez les méthodes `dropByName` ou `dropByIndex` de la collection `lesTables`.

Enfin, notons que chaque objet table permet d'accéder à la collection des index et des relations. Nous ne rentrerons pas dans le détail du service `com.sun.star.sdbcx.Index`, mais retenons que, au même titre que la collection `Tables`, cette collection permet d'accéder aux champs indexés par l'intermédiaire d'une collection `columns`. La collection

permet, suivant ce qui vient d'être décrit, d'ajouter ou d'enlever des index. Chaque index expose entre autres les propriétés `IsUnique` et `IsPrimaryKeyIndex`.

Les requêtes pré-enregistrées

La connexion établie donne accès à la collection des requêtes enregistrées dans la source de données, les `Queries`. Ainsi, plutôt que créer des requêtes par macro vous pouvez directement accéder aux requêtes créées dans le gestionnaire de sources de données. Afin d'éviter toute ambiguïté, nous garderons le terme de `Query` pour ce type de requête, réservant le mot *requête* à la manipulation de véritables requêtes SQL (comme chaînes de caractères) dans les macros. Nous allons lire et modifier la collection des requêtes qui y sont présentes.

```
rem Code17-02.sxw  bibli : BaseDonnees Module5
Option Explicit

Sub InfoQueries()
dim LesQueries As Object, uneQuery As Object
dim i As Long, NombreQueries As Long, chaine As String

ConnecterSource
'Collection des requêtes de la source de données
LesQueries = maConnexion.Queries
NombreQueries = LesQueries.Count
print "Nombre de Requetes : " & NombreQueries

for i = 0 to NombreQueries -1 'Boucle sur les requêtes
  uneQuery = LesQueries.getByName(LesQueries.ElementNames(i))
  chaine = "Requête : " & uneQuery.Name
  chaine = chaine & chr(13) & uneQuery.Command
  MsgBox(chaine)
next i
DeconnecterSource
End Sub
```

Nous constatons que la macro `InfoQueries` ressemble beaucoup à la macro `InfoTables`. En fait, `LesQueries` est une collection et donc toutes les méthodes de manipulation évoquées précédemment sont utilisables (`count`, `hasByName`, `getByName`, `dropByName`...)

L'objet de base est `uneQuery` du type `com.sun.star.sdb.QueryDefinition`. Les propriétés `Name` et `Command` suffisent à le définir. Notez que nous avons récupéré chaque `Query` à partir de son nom, ceci pour contourner une bogue dans l'accès par index.

Enregistrer une requête

Pour ajouter une nouvelle requête à la collection, nous n'aurons pas besoin d'ouvrir une connexion. Le codage présenté ici est compatible avec OpenOffice.org 2.0 et les versions antérieures.

```
rem Code17-02.sxw  bibli : BaseDonnees Module6
Option Explicit

Sub CreerRequete()
Dim monDbContext As Object, maSource As Object
Dim defQueries As Object, descrQuery As Object, dbDoc As Object
Const nomSource = "BaseExemple"
Const nomQuery = "Clients nés après 1950"

monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource = monDbContext.getByName(NomSource)
defQueries = maSource.QueryDefinitions

if not defQueries.hasByName(nomQuery) then
  'création d'un nouvelle requête par descripteur
  descrQuery = defQueries.createInstance()
  descrQuery.Command = "SELECT * FROM client WHERE NAISS > 1950"
  defQueries.insertByName(nomQuery, descrQuery)
  on Error goto OOoV1
  dbDoc = maSource.DatabaseDocument 'erreur avant OOo 2.0 !
  on Error Goto 0 ' supprimer le traitement d'erreur
  dbDoc.store ' sauver la requête dans le fichier .odb
  Goto L1
  OOoV1:
  Resume L1  ' igorer l'erreur : OOo 1.1 ou inférieur
  L1:
  on Error Goto 0
  MsgBox("Requête enregistrée : " & nomQuery)
else
  print "Nom déjà existant - Impossible de créer la requête"
endif

End Sub
```

Nous récupérons l'objet source de données, dont la propriété `QueryDefinitions` expose un conteneur de toutes les requêtes existantes. Après avoir vérifié qu'il n'existe pas déjà une requête du même nom, nous demandons un descripteur de requête vierge de la forme `com.sun.star.sdb.QueryDescriptor`. Le SDK décrit les différentes propriétés du descripteur ; habituellement, il suffit de remplir la propriété `Command` avec l'instruction SQL de la requête. La méthode `insertByName` du conteneur de requêtes permet d'y insérer notre requête.

Le travail est terminé pour OpenOffice.org 1.1.x. Avec la version 2.0 en revanche, il est nécessaire de sauvegarder le conteneur de requêtes dans le document `.odb` associé à la base de données. Nous récupérons ce document par la propriété `DatabaseDocument` de la source de données. Cette propriété existe seulement à partir de la version 2.0. Pour que notre codage fonctionne aussi avec les versions précédentes, nous avons ajouté un traitement d'erreur : si l'instruction récupérant le document `.odb` déclenche une erreur, la sauvegarde est évitée. Nous avons rétabli le traitement d'erreur normal dès l'instruction suivante afin de détecter toute anomalie de sauvegarde.

Cette technique pourra être efficacement utilisée lors de migration de bases de données MS-Access, ainsi que nous le verrons à la fin de ce chapitre.

Modifier une requête

Il est parfois utile de modifier l'instruction SQL d'une requête existante afin de rechercher des enregistrements dépendant des besoins du moment. Avec la version 2.0 d'OpenOffice.org, on peut créer avec l'interface utilisateur un état basé sur le résultat d'une requête pré-enregistrée. Ce même état peut sortir d'autres sélections d'enregistrement, simplement en modifiant l'instruction SQL de la requête. Or ceci est facile à réaliser par programmation en s'inspirant du codage précédent. Deux instructions sont nécessaires :

```
descrQuery = defQueries.getByName(nomQuery)
descrQuery.Command = "SELECT * FROM client WHERE NAISS < 1995"
```

Nous avons constaté qu'il n'est même pas nécessaire avec la version 2.0 d'OpenOffice.org de sauvegarder le document `.odb`.

Ouvrir un rapport (ou état)

La version 2.0 d'OpenOffice.org permet, à l'instar de MS-Access, de mémoriser un état dans le document `.odb` lié à la source de données. Voici comment afficher le résultat d'un état et faire une copie de ce résultat.

```
rem Code17-04.odt  bibli : Standard Module1
Option Explicit

Dim astuce As Object ' pour garder l'état ouvert après la fin de la macro

Sub AfficherEtat()
Dim monDbContext As Object, maSource As Object, maConnexion As Object
Dim dbDoc As Object, monEtat As Object, lesEtats As Object
Dim OdbOpt(0) as new com.sun.star.beans.PropertyValue
Dim stdOpenArgs(0) as new com.sun.star.beans.PropertyValue
Dim adrOdb As String, adrEtat As String
Const nomSource = "usine", nomEtat = "ProdPrixSup10"
```

```
monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource = monDbContext.getByName(NomSource)
dbDoc = maSource.DatabaseDocument
adrOdb = dbDoc.URL
OdbOpt(0).Name = "Hidden"
OdbOpt(0).Value = True
dbDoc = StarDesktop.loadComponentFromURL(adrOdb, "_blank", 0, OdbOpt())
lesEtats = dbDoc.ReportDocuments

' contournement de bogue : une des instructions suivantes
' est indispensable pour visualiser l'état
if MsgBox("Voir encore l'état après la fin de macro ?", 4) = 6  then
  ' Réponse OUI : affichage maintenu après la fin de la macro
  ' le résultat de .getByName doit être mémorisé
  ' dans une variable (inutilisée) hors de la macro !
  astuce = lesEtats.getByName(nomEtat)
else  ' Réponse NON : affichage seulement pendant la macro
  ' Notez que le résultat de .getByName n'est pas utilisé !
  lesEtats.getByName(nomEtat)
end if

stdOpenArgs(0).Name = "ActiveConnection"
maConnexion = dbDoc.Datasource.getConnection("","")
stdOpenArgs(0).Value = maConnexion
monEtat = lesEtats.loadComponentFromURL(nomEtat, "_blank", 0, stdOpenArgs())
MsgBox("Regardez l'état")
' Créer un document texte, copie de l'état actuel
adrEtat = ConvertToURL("C:\Docs OpenOffice\PrixSuperieur10.odt")
monEtat.storeAsURL(adrEtat, array())

dbDoc.close(True)
maConnexion.close
maConnexion.dispose
End Sub
```

D'abord, nous devons charger le document `.odb` qui contient (entre autres) les états. Pour cela, il nous faut son adresse URL, que nous obtenons à partir de la propriété `DatabaseDocument` exposée par la source de données. Ici, nous avons choisi de charger le document sans l'afficher. Le conteneur des états est récupéré par la propriété `ReportDocuments`.

Maintenant, nous allons contourner une bogue de la version OpenOffice.org que nous utilisons pour ce livre. Sans précautions, l'état apparaît quelques dixièmes de secondes seulement. Cette partie est susceptible de changer en fonction des futures corrections d'OpenOffice.org. Notre codage donne deux solutions, l'une limitée à l'affichage pendant l'exécution de la macro, l'autre donnant un affichage persistant après exécution.

Ensuite, nous chargeons, de manière visible, l'état à partir de son conteneur. Notez la propriété `ActiveConnection` : nous devons auparavant effectuer une connexion à la base

de données pour obtenir la mise à jour de l'état en fonction des données réelles de la base. L'affichage peut prendre quelques secondes pour obtenir le résultat final. Nous pouvons ensuite effectuer une copie de sauvegarde de l'état avec les valeurs présentes. Il ne nous reste plus qu'à fermer le document `.odb` ainsi que la connexion.

Accéder aux données avec le langage SQL

La raison d'être d'une base de données est de consulter et manipuler les données qu'elle contient. Les données sont ainsi vivantes et le langage SQL nous permet d'extraire des constructions simples ou complexes de ces données. Cette section va nous permettre d'illustrer l'emploi du langage SQL au travers d'OpenOffice.org. À titre d'exemple, nous utiliserons la source de données `BaseExemple` fournie, tout en étant conscients de ses limitations, notamment sur les jointures qui ne sont pas possibles avec ce format. Néanmoins, ceci ne restreint en rien ce qui est dit. Utiliser les jointures ne relève que d'un changement de syntaxe SQL et du moment que la base les accepte, OpenOffice.org n'y verra aucun inconvénient.

> **AVERTISSEMENT Usage des bases de données**
>
> Les exemples donnés sont volontairement simplifiés. Ne vous lancez pas dans la définition ou la programmation d'une base de données à *usage professionnel* si vous n'êtes pas compétent dans ce domaine. L'économie que vous pourriez espérer réaliser risque de vous coûter très cher.

Exploiter les résultats d'une requête

Nous allons utiliser la table `client` de notre source de données `BaseExemple`. Notre requête SQL va consister à obtenir une image de la table, ordonnée selon les valeurs croissantes du champ `ID`. En effet, les enregistrements de la table n'ont pas d'ordre particulier.

Cette macro est assez longue, mais peu complexe. Nous la commenterons derrière ce codage.

```
rem Code17-02.sxw  bibli : avecSQL Module1
Option Explicit

Sub LireResultatRequeteSQL()
dim maRequete As Object, resuQuery As Object
dim instrSQL As String, monSignet As Variant
dim execOK As Boolean, info As String, cr As String
cr = chr(13) ' retour à la ligne, pour les messages

ConnecterSource
' Texte de la requête
```

```
instrSQL = "select * from client order by ID"
' Envoyer la requête
maRequete = maConnexion.createStatement()
resuQuery = maRequete.executeQuery(instrSQL)
' Traitement du résultat
With resuQuery
  execOK = .next ' obtenir la première ligne
  GoSub afficher
  execOK = .next ' ligne suivante
  GoSub afficher
  execOK = .previous ' ligne précédente, donc la première ligne
  GoSub afficher
  execOK = .last ' obtenir la dernière ligne
  GoSub afficher
  execOK = .relative(-3) ' remonter de 3 lignes
  GoSub afficher
  execOK = .absolute(3) ' aller à la ligne 3
  GoSub afficher
  monSignet = .Bookmark  ' mémoriser cette position (ligne 3)

  execOK = .relative(-9999) ' remonter au-delà du début !
  GoSub afficher
  ' revenir à la position du signet
  execOK = .moveToBookmark(monSignet)
  GoSub afficher
  execOK = .absolute(9999) ' descendre au-delà de la fin !
  GoSub afficher
  execOK = .first ' retourner à la première ligne
  GoSub afficher
  ' aller 3 lignes plus bas que la position du signet
  execOK = .moveRelativeToBookmark(monSignet, 3)
  GoSub afficher

  DeconnecterSource
  Exit Sub ' fin du sous-programme

  afficher: ' --- sous-programme interne ---
  if execOK then
    info = "Ligne : " & .Row  & "   " & _
           "Client numéro = " & .Columns(0).Short & cr & _
           "Nom = " & .Columns.getByName("NOM").String & cr & _
           "Prénom = " & .Columns(2).String & cr & _
           "Année de naissance : " & .Columns(3).Int
    if .isFirst then _
           info = info & cr & "Curseur en première ligne"
    if .isLast  then _
           info = info & cr & "Curseur en dernière ligne"
  else
    info = "Déplacement non exécuté !"
    if .isBeforeFirst then _
           info = info & cr & "Curseur au début"
```

```
    if .isAfterLast   then _
            info = info & cr & "Curseur à la fin"
  end if
  MsgBox(info)
  Return
End With
End Sub
```

Après nous être connectés à la source de données, nous utilisons la fonction `createStatement` pour obtenir l'objet `maRequete` qui sera capable d'effectuer une requête SQL : il suffit d'utiliser sa fonction `executeQuery` , qui prend pour argument l'instruction SQL et renvoie un résultat.

Le résultat de la requête, `resuQuery`, est un objet offrant le service `com.sun.star.sdb.ResultSet`. Le reste de la macro va vous montrer les possibilités de cet objet. Il faut savoir que le résultat d'une requête a une structure logique en lignes et colonnes ; chaque ligne est un enregistrement, chaque colonne est un champ du résultat. L'intérêt d'un `ResultSet` est sa capacité à explorer les lignes du résultat de la commande SQL.

Il existe trois services appelés `ResultSet`. Ils sont définis dans les branches de l'API :

- `com.sun.star.sdb`
- `com.sun.star.sdbcx`
- `com.sun.star.sdbc`

Le résultat d'une requête SQL est un objet `ResultSet` exposant le service `sdb`, qui inclut le service `sdbcx`, qui inclut lui-même `sdbc`. Il s'agit donc de la variante la plus élaborée.

Donnons deux explications sur le codage employé dans la macro :

- Dans la zone encadrée par `With resuQuery` et `End With`, le mot `resuQuery` est sous-entendu quand un terme commence par un point. Par exemple, `.next` doit être interprété par OOoBasic comme `resuQuery.next`. Cela allège considérablement l'écriture (et la relecture).
- Nous utilisons un sous-programme interne appelé par `GoSub`. Lisez le chapitre 7 si vous ne connaissez pas cette structure.

Sitôt après l'exécution de la requête, l'objet `resuQuery` est positionné « avant le début » des résultats. La fonction `next` avance d'une ligne, un enregistrement, dans la lecture du résultat. Donc, suite au premier `next`, nous sommes sur la première ligne (si résultat il y a). La fonction `next`, comme celles que nous verrons ensuite, renvoie une valeur `True` si le déplacement a pu être réalisé, `False` dans le cas contraire.

Le sous-programme interne `afficher` construit une chaîne de caractères de plusieurs lignes, puis l'affiche. Il récupère le rang de la ligne courante dans la propriété `Row` du

ResultSet. Attention : ce rang ne concerne que le résultat de la requête, ce n'est pas le rang dans la table client.

Chaque champ (ou colonne) du résultat est accessible par le ResultSet. La propriété Columns est indexée par le rang du champ, numéroté à partir de zéro de gauche à droite. Comme notre résultat reprend la structure de la table client, le nombre et la signification des champs seront les mêmes :

- rang 0 : champ ID ;
- rang 1 : champ NOM ;
- rang 2 : champ PRENOM ;
- rang 3 : champ NAISS ;
- rang 4 : champ SITFAM.

La valeur du champ (ou contenu de la colonne pour la ligne en cours) est obtenue avec une fonction dont le nom dépend du type de donnée. En réalité, nous utilisons une des nombreuses fonctions getXXX d'un ResultSet, mais OOoBasic nous dispense du get. Il existe une fonction getXXX pour chacun des types de résultats possibles (tableau 17-5), mais OOoBasic ne possède pas tous les types de données nécessaires pour les représenter.

Tableau 17–5 Fonctions getXXX d'un ResultSet

Fonction	Type Basic	Fonction	Type Basic
getByte	Integer	getBytes	Array(Integer)
getShort	Integer	getdate	Object
getInt	Long	getTime	Object
getLong	<inexistant>	getTimestamp	Object
getFloat	Single	getCharacterStream	Object
getDouble	Double	getBinaryStream	Object
getBoolean	Boolean	getObject	Object
getString	String		

Les fonctions getXXX sont capables de convertir la donnée réelle dans leur propre type. Ainsi, un entier de faible valeur pourrait être récupéré avec getShort, getFloat, getDouble, ou même getByte ou getString.

Dans notre cas, le numéro d'identification du client est obtenu avec la fonction getShort, abrégée en Short, le nom et le prénom sont obtenus avec getString, abrégé en String, l'année de naissance est obtenue par getInt, abrégé en Int.

La récupération de la valeur du champ NOM montre une meilleure manière d'accéder à la bonne colonne : on utilise la fonction getByName de l'objet Columns de l'objet ResultSet.

En résumé, on peut obtenir la valeur d'une colonne d'un `ResultSet` de trois manières. Pour un champ texte :

```
valeur = resuQuery.Columns.getByName("PRENOM").String
valeur = resuQuery.Columns(2).String
valeur = resuQuery.getString(3)
```

Pour un champ numérique, on utilisera `Double` ou `Int`, etc. selon le type numérique du champ :

```
valeur = resuQuery.Columns.getByName("NAISS").Double
valeur = resuQuery.Columns(3).Double
valeur = resuQuery.getDouble(4)
```

Vous aurez remarqué le décalage de numérotation de colonne entre la deuxième et la troisième méthode.

Un champ `Date` est vu dans l'API comme une structure `com.sun.star.util.date`. Cette structure se compose des sous-éléments :

- `Day` : le jour entre 1 et 31
- `Month` : le mois entre 1 et 12
- `Year` : l'année, exemple 1975

Pour remplir un champ `Date`, il faut au préalable définir et remplir une variable objet :

```
Dim nouvDate As New com.sun.star.util.date
nouvDate.Year = 1975
nouvDate.Month = 12
nouvDate.Day = 31
```

La syntaxe SQL pour rechercher des enregistrements d'une date donnée dépend du moteur de base de données : relisez la documentation de l'éditeur du moteur que vous utilisez. Une autre possibilité est d'effectuer une requête avec le gestionnaire de données, puis d'éditer la requête en désactivant le mode Ébauche, pour voir l'instruction SQL qu'elle avait créée.

Le `ResultSet` est aussi capable de nous indiquer si la ligne courante est la première ou la dernière du résultat, avec les fonctions respectives `isFirst` et `isLast`, qui renvoient une valeur booléenne.

Revenons à la séquence principale de la macro. Le `next` suivant nous permet d'afficher la deuxième ligne. La fonction `previous`, elle, remonte à la ligne précédente. Les fonctions `first` et `last` nous positionnent respectivement sur la première ou la dernière ligne du `ResultSet`. Il est encore possible de se déplacer en relatif (descendre ou remonter de n lignes par rapport à la position actuelle), ou en absolu (aller à la ligne numéro x du

résultat). Le `ResultSet` est aussi capable de mémoriser le rang de la ligne courante, avec un signet (anglais : `Bookmark`). Notez que la variable de signet doit être du type `Variant`.

Nous utilisons les déplacements relatif et absolu pour demander un déplacement impossible. Effectivement, dans ces deux cas, `execOK` recevra la valeur `False` et le sous-programme `afficher` exécutera la partie `else` de son test. Le `ResultSet` nous fournit encore deux autres informations utiles : `isBeforeFirst` renvoie `True` si nous sommes rendus avant la première ligne, `isAfterLast` renvoie `True` si nous sommes après la dernière ligne. Notez que si la requête n'avait donné aucun résultat, ces deux fonctions renverraient simultanément `True`, et bien sûr nous ne pourrions nous déplacer.

À la place d'une demande de déplacement impossible à réaliser, nous aurions pu utiliser les fonctions `beforeFirst` ou `afterLast`, qui servent à se positionner avant la première ou après la dernière ligne.

Le `ResultSet` offre deux fonctions pour nous déplacer par rapport à un signet mémorisé : `moveToBookmark` va directement à la position de celui-ci, `moveRelativeToBookmark` cherche une position relative au signet (avant ou après lui).

Autres exemples de requêtes d'interrogation

D'autres interrogations sont possibles en utilisant des instructions SQL adéquates (et dans une syntaxe acceptée par le pilote de la base de données). Voici quelques exemples basiques.

```
select count(*) as nb from client
```

Renvoie au maximum une ligne, contenant dans une colonne le nombre d'enregistrements de la table client.

```
select libelle, prix from produits where prix > 2.99 order by prix asc
```

Recherche dans la table `produits` ceux dont le prix est supérieur à 2,99 et renvoie le résultat en indiquant le libellé et le prix, mis par ordre de prix croissant.

Insérer un enregistrement dans une table

Avec l'instruction SQL

L'accès aux données est également possible en écriture. La requête SQL utilisée pour insérer un enregistrement est un `INSERT`. Cependant, une autre méthode de l'objet requête sera utilisée : `executeUpdate`.

Nous allons ajouter une entrée supplémentaire dans la table `produit`. Le codage sera plus complexe qu'à l'accoutumée, mais nous l'expliquerons pas à pas.

```
rem Code17-02.sxw  bibli : avecSQL Module2
Option Explicit

Sub InsererDonneesParSQL()
dim maRequete As Object, resuQuery As Object
dim LeLibelle As String, LePrix As Double, instrSQL As String
dim nbLignesEcrites As Long, indexP As Long, x As Long
'Demande des renseignements à l'utilisateur
if nouvelArticle(LeLibelle, LePrix) then
  ConnecterSource
  'chercher la valeur maximale de ID_PRODUIT
  maRequete = maConnexion.createStatement()
  resuQuery = maRequete.executeQuery("select ID_PRODUIT from produits")
  indexP = 1
  while  resuQuery.next
    x = resuQuery.Columns.getByName("ID_PRODUIT").Int
    if x >= indexP then indexP = x +1
  wend

  'construction de la requête d'insertion
  instrSQL = "Insert into produits " & _
             "(PRIX, LIBELLE, ID_PRODUIT) values(" & _
             NombreAnglais(LePrix) & "," & _
             Quote(LeLibelle) & "," & _
             NombreAnglais(indexP) & ")"
  'ajout du nouvel enregistrement
  nbLignesEcrites = maRequete.executeUpdate(instrSQL)
  MsgBox("Instruction SQL :" & chr(13) & instrSQL & chr(13) & _
         "Nombre de lignes écrites : " & nbLignesEcrites)
  DeconnecterSource
end if
End Sub

Function nouvelArticle(nom As String, prix As Double) As Boolean
Dim Dlg As Object, bibli As Object
Dim monDialogue As Object, exitOK As String
exitOK = com.sun.star.ui.dialogs.ExecutableDialogResults.OK
bibli = DialogLibraries.GetByName("avecSQL")
monDialogue = bibli.GetByName("DialProduit")
Dlg = CreateUnoDialog(monDialogue)
if Dlg.Execute = exitOK then
  nouvelArticle = true
  nom = Dlg.getControl("Libelle").Text
  prix = Dlg.getControl("Prix").Value
else
  nouvelArticle = false
end if
End Function
```

```
'transforme la virgule en point décimal
Function NombreAnglais(ByVal txtNombre As String) As String
Dim x As Long
x = Instr(txtNombre, ",")
if x > 0  then Mid(txtNombre, x, 1, ".")
QuoteNombre = txtNombre
End Function
'double les quotes de l'argument chaine
'puis l'encadre entre quotes
Function Quote(ByVal chaine As String) As String
Dim i As Long
i = instr(chaine,"'")
while i>0
  chaine = left(chaine,i) & "'" & mid(chaine,i+1)
  i = i+2
  i = instr(i,chaine,"'")
wend
Quote = "'" & chaine & "'"
End Function
```

Nous utilisons une petite boîte de dialogue pour demander à l'utilisateur le descriptif du produit à ajouter : libellé et prix. L'intérêt de la boîte de dialogue est de mieux contrôler les données utilisateur. La boîte est utilisée par la fonction `nouvelArticle` qui, soit renvoie une valeur `False` si l'utilisateur a changé d'avis, soit renvoie la valeur `True` si l'utilisateur a fourni les renseignements.

Dans ce dernier cas, `LeLibelle` contient un texte et `LePrix` contient un nombre réel. Il nous manque une information, qui est le numéro d'identité du produit. Ce numéro doit être unique. Dans une base de données plus professionnelle que dBase, nous aurions un champ auto-incrémenté par la base elle-même et nous n'aurions pas à nous en préoccuper. Ici, nous allons choisir une valeur supérieure à toutes les valeurs d'identité déjà existantes dans la table. Pour cela, nous allons effectuer une première requête SQL.

Avec un pilote de base de données un peu plus évolué que celui fourni avec OpenOffice.org, il suffirait de faire une requête demandant le `MAX()` de la colonne `ID_PRODUIT`. Ici, nous devons faire le travail nous-mêmes. Nous effectuons une requête qui retourne la valeur de ce champ dans toutes les lignes existantes. Le résultat de la requête est exploré complètement avec des appels successifs de la méthode `next`. À la fin de la boucle `while`, la variable `indexP` contient une valeur supérieure d'une unité à la valeur maximale trouvée.

L'étape suivante consiste à construire la requête SQL qui servira à insérer le nouvel enregistrement, sous forme d'une chaîne de caractères. Trois difficultés sont à résoudre afin de satisfaire la syntaxe du langage SQL :

1. Une valeur de texte doit être entourée d'apostrophes simples.
2. Si une valeur de texte contient une apostrophe simple, elle doit être doublée d'une autre.

3 Les nombres avec une partie décimale doivent utiliser le point comme séparateur décimal, et non la virgule comme en français.

Les deux premières sont traitées par la fonction `Quote`. Remarquez le terme `ByVal`, qui sert à récupérer une copie de la valeur de l'argument de la fonction. Ceci nous permet de modifier localement la variable `chaine` sans modifier la variable originale. La boucle `while` repère et double les apostrophes existant dans le texte. La dernière instruction sert à renvoyer en résultat le texte encadré d'apostrophes.

La troisième difficulté est résolue avec la fonction `NombreAnglais`. Nous l'utilisons en lui transmettant un nombre, mais comme l'argument attendu est une chaîne de caractères, OOoBasic se charge de la conversion en texte. Ensuite, il suffit de rechercher si ce texte contient une virgule et de la remplacer par un point.

Ayant fabriqué l'instruction SQL, nous la transmettons au pilote de la base de données avec la fonction `executeUpdate`, qui renvoie le nombre d'enregistrements écrits. Dans notre cas, ce sera un seul, si tout se passe bien.

PIÈGE Base multi-utilisateurs

L'exemple utilisé peut engendrer des incohérences dans le cas d'une base de données multi-utilisateurs. Il faudrait utiliser une transaction pour couvrir l'ensemble des deux requêtes, comme nous le verrons plus loin. La deuxième version de l'exemple comporte aussi le même problème potentiel.

Avec le ResultSet

Le `ResultSet` est capable de modifier le contenu de la table initiale. Nous allons refaire l'exemple précédent, en tirant partie de ces capacités. Ce sera bien plus simple.

```
rem Code17-02.sxw  bibli : avecSQL Module3
Option Explicit

Sub InsererDonneesParResultSet()
dim maRequete As Object, resuQuery As Object
dim LeLibelle As String, LePrix As Double, instrSQL As String
dim nbLignesEcrites As Long, indexP As Long, x As Long

'Demande des renseignements à l'utilisateur
if nouvelArticle(LeLibelle, LePrix) then
  ConnecterSource
  'chercher la valeur maximale de ID_PRODUIT
  maRequete = maConnexion.createStatement()
  resuQuery = maRequete.executeQuery("select * from produits")
  indexP = 1
  while  resuQuery.next
    x = resuQuery.Columns.getByName("ID_PRODUIT").Int
    if x >= indexP then indexP = x +1
  wend
```

```
  'ajout du nouvel enregistrement
  resuQuery.moveToInsertRow ' aller à la zone de préparation
  resuQuery.updateDouble(3, LePrix) ' troisième colonne
  resuQuery.updateString(2, LeLibelle)
  resuQuery.updateInt(   1, indexP) ' première colonne
  resuQuery.insertRow ' insérer l'enregistrement dans la table
  DeconnecterSource
end if
End Sub
```

La première partie est identique à l'exemple utilisant SQL. Ici cependant, notre instruction SQL va récupérer toute la table `produits`. Le résultat de la requête SQL nous servira pour ajouter un enregistrement.

Dans un premier temps, nous nous positionnons dans une zone spéciale, la ligne d'insertion, grâce à la méthode `moveToInsertRow`. Puis nous allons remplir chaque champ du nouvel enregistrement. Enfin la ligne d'insertion complétée est insérée physiquement dans la table avec la méthode `insertRow`.

Le remplissage des champs utilise des méthodes `updateXXX` qui sont le pendant des fonctions `getXXX` déjà vues. Le premier argument est le rang du champ dans l'enregistrement. Attention, ici *la numérotation commence à 1*, alors qu'en lecture d'un `ResultSet`, la numérotation des colonnes part de zéro !

Nous ne l'avons pas employé ici, mais, après avoir inséré un ou plusieurs enregistrement(s), la méthode `moveToCurrentRow` permet de revenir où on en était dans le `ResultSet`.

Modifier un enregistrement d'une table

Avec l'instruction SQL

La requête SQL utilisée sera un `UPDATE`. Attention à cette instruction, qui peut tout détruire dans votre table. La requête sera exécutée avec la méthode `executeUpdate`. Nous ne donnerons pas d'exemple car le pilote SDBC d'OpenOffice.org ne semble pas fonctionner correctement sur cette instruction.

Avec un ResultSet

Pour simplifier l'exemple, nous allons appliquer une hausse de 25 % au prix du premier article que nous trouverons dans la table `produits`.

```
rem Code17-02.sxw  bibli : avecSQL Module4
Option Explicit

Sub ModifierDonneesParResultSet()
```

```
dim maRequete As Object, resuQuery As Object
dim instrSQL As String
dim unPrix As Double, LeLibelle As String
ConnecterSource
' Texte de la requête
instrSQL = "select * from produits"
' Envoyer la requête
maRequete = maConnexion.createStatement()
resuQuery = maRequete.executeQuery(instrSQL)
if resuQuery.next then
  LeLibelle = resuQuery.Columns(1).String
  unPrix = resuQuery.Columns(2).Double
  unPrix = unPrix * 1.25 ' hausse de 25%
  resuQuery.updateDouble(3, unPrix)
  resuQuery.updateRow ' transférer dans la table
  MsgBox("Produit modifié : " & LeLibelle)
end if
DeconnecterSource
End Sub
```

Après exécution de la requête, nous utilisons la fonction `next` pour nous positionner sur la première ligne (si le résultat de la requête est vide, on ne fera rien). Nous récupérons le libellé et le prix de l'article. Après application de la hausse, nous utilisons `updateDouble` pour mettre à jour le champ. Attention, ceci ne modifie pas la table, car nous travaillons sur une zone tampon. Nous pourrions modifier ainsi d'autres champs. Puis nous transférons la totalité de la ligne modifiée dans la table avec la méthode `updateRow`.

Dans une application plus complexe, vous pouvez avoir besoin d'annuler des modifications en cours et recommencer tous les `updateXXX`. Ceci est possible *avant* d'exécuter `updateRow` ; il suffit d'employer la méthode `cancelRowUpdates`.

Effacer des enregistrements

En utilisant l'instruction SQL

L'exemple suivant supprime tous les enregistrements de la table `produits` dont le prix est inférieur à 1 en utilisant une instruction SQL `delete`. Ne pas mettre de clause `where` reviendrait à vider complètement la table. Nous utilisons la fonction `executeUpdate`, comme précédemment.

```
rem Code17-02.sxw  bibli : avecSQL Module5
Option Explicit

Sub EffacerDonneesParSQL()
dim maRequete As Object, instrSQL As String
dim nbLignesTraitees As Long
```

```
ConnecterSource
'Envoi de la requête
instrSQL = "delete from produits where prix < 1"
maRequete = maConnexion.createStatement()
nbLignesTraitees = maRequete.executeUpdate(instrSQL)

MsgBox("Nombre de lignes traitées : " & nbLignesTraitees)
DeconnecterSource
End Sub
```

En utilisant le ResultSet

L'exemple précédent serait plus complexe à réaliser avec le `ResultSet`. À la place, nous présenterons la suppression d'un enregistrement particulier. Pour simplifier, nous supprimerons l'enregistrement de rang 2 dans le `ResultSet`.

```
rem Code17-02.sxw  bibli : avecSQL Module6
Option Explicit

Sub EffacerDonneesParResultSet()
dim maRequete As Object, resuQuery As Object
dim instrSQL As String, LeLibelle As String

ConnecterSource
' Texte de la requête
instrSQL = "select * from produits"
' Envoyer la requête
maRequete = maConnexion.createStatement()
resuQuery = maRequete.executeQuery(instrSQL)
if resuQuery.absolute(2) then
  LeLibelle = resuQuery.Columns(1).String
  resuQuery.deleteRow 'supprimer de la table
  MsgBox("Produit supprimé : " & LeLibelle)
end if
DeconnecterSource
End Sub
```

Les différentes capacités des ResultSet

Dans nos exemples, nous utilisons un `ResultSet` avec ses possibilités par défaut. Celles-ci sont des propriétés de l'objet requête. La propriété `ResultSetType` possède une valeur parmi trois (constantes nommées) :

```
com.sun.star.sdbc.ResultSetType.FORWARD_ONLY
com.sun.star.sdbc.ResultSetType.SCROLL_INSENSITIVE
com.sun.star.sdbc.ResultSetType.SCROLL_SENSITIVE
```

La première valeur n'autorise que les déplacements vers les lignes suivantes. La deuxième valeur est celle utilisée par défaut avec notre pilote. La différence entre la deuxième et la troisième est qu'un `ResultSet SCROLL_SENSITIVE` se met à jour des modifications effectuées par d'autres utilisateurs de la base de données.

La propriété `ResultSetConcurrency` possède une valeur parmi deux :

```
com.sun.star.sdbc.ResultSetConcurrency.READ_ONLY
com.sun.star.sdbc.ResultSetConcurrency.UPDATABLE
```

La première valeur indique que le résultat est en lecture seule, alors que le deuxième permet de modifier. La deuxième valeur est celle utilisée par défaut avec notre pilote.

Si on souhaite une valeur particulière, les propriétés `ResultSetType` et `ResultSetConcurrency` doivent toutes deux être initialisées avant d'exécuter l'instruction SQL.

Les valeurs nulles

Un champ d'une table peut accepter une valeur `Null`, signifiant « absence de valeur », ce qui est différent d'une valeur zéro ou d'une chaîne de caractères vide. Après avoir lu la valeur d'une colonne de `ResultSet` avec une fonction `getXXX`, l'ambiguïté est levée en testant la propriété `wasNull`. Comme vous l'avez deviné, elle vaut `True` si le dernier `getXXX` a récupéré une valeur `Null`.

Pour « remplir » une colonne avec une valeur `Null`, employez la méthode `updateNull` au lieu d'un `updateXXX`.

Les requêtes paramétrées

Lorsque vous avez à insérer un grand nombre d'enregistrements dans une table, ou à effectuer des modifications avec de nombreuses instructions SQL, l'utilisation d'une requête paramétrée est recommandée car elle réduit la charge de travail du pilote de la base de données.

Une requête paramétrée est une instruction SQL dans laquelle certaines valeurs sont remplacées par un point d'interrogation. Dans un premier temps, l'instruction SQL est envoyée au pilote de la base de données. Il en effectue l'analyse syntaxique et mémorise une version compilée de l'instruction, puis il renvoie une référence sur cette instruction.

Dans un deuxième temps, à chaque insertion ou modification à réaliser, la référence est envoyée au pilote accompagnée des valeurs manquantes. Il peut alors éviter la phase d'analyse de l'instruction SQL.

Le mécanisme de requête paramétrée est pris en charge par l'objet connexion. Dans cet exemple, nous ajoutons 50 enregistrements à la table `stocks`. Le contenu est sans signification, seul le principe est intéressant.

```
rem Code17-02.sxw  bibli : avecSQL Module7
Option Explicit

Sub InsererDonneesParRequetePreparee()
dim maRequete As Object
dim LeLibelle As String, LePrix As Double, instrSQL As String
dim x As Long

ConnecterSource
maRequete = maConnexion.prepareStatement( _
  "INSERT INTO stocks (REF, DISPO) VALUES(?, ?)")
for x = 1 to 50
  maRequete.setString(1, "Type" & x)
  maRequete.setInt(2, CLng(rnd*10000))
  maRequete.execute
next
maRequete.dispose
DeconnecterSource
End Sub
```

Les variables SQL sont remplies avec des instructions `setXXX` dont le premier argument est le rang (débutant à 1) du point d'interrogation dans l'instruction SQL, et le deuxième argument est la valeur elle-même. Dans notre cas, la première variable doit être du texte, la deuxième est un entier. Il existe autant de méthodes `setXXX` que de méthodes `getXXX` ou `updateXXX`, y compris le `setNull`.

La valeur d'une variable SQL est conservée jusqu'à une nouvelle affectation. La méthode `clearParameters` permet de réinitialiser toute modification des paramètres.

Accéder aux données avec un RowSet

Nous venons d'explorer une méthode de manipulation des données qui découlaient d'une requête en langage SQL. L'API d'OpenOffice.org propose également un autre objet, le `RowSet` issu du service `com.sun.star.sdb.RowSet`.

Un `RowSet` possède toutes les possibilités d'un `ResultSet`, mais il intègre en plus un objet requête SQL et de nouvelles fonctionnalités. C'est cette approche que nous allons maintenant explorer.

Créer un RowSet

Pour qu'un `RowSet` soit utile, il doit être connecté à une source de données. Il existe deux manières de le faire ; la première est similaire à la connexion d'une source de données.

```
rem Code17-02.sxw  bibli : avecRowSet Module1
Option Explicit

Sub CreerRowSetAutonome()
dim unRowSet As Object

UnRowSet=createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  .dataSourceName = "BaseExemple"
  .user = ""
  .password = ""
  ' préciser l'origine des données
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = "client"
  .execute ' effectuer la requête implicite
  ' analyser le résultat avec le RowSet intégré
  .next ' afficher le premier enregistrement
  MsgBox("Nom = " & .Columns.getByName("NOM").String)

  .dispose ' détruire le RowSet
End With
End Sub
```

Nous commençons donc par créer un objet `UnRowSet` vierge en utilisant le service `com.sun.star.sdb.RowSet`. La propriété `dataSourceName` est initialisée avec le nom de la source de données, et les paramètres de connexion nécessaires sont mis dans les propriétés `user` et `password`. En ce sens, le `RowSet` nous affranchit de tout le travail préliminaire de définition du `dataBaseContext`. La tâche à accomplir par le `RowSet` est précisée avec deux propriétés : `CommandType`, de type `Long`, et `Command`, de type `String`. La première reçoit une constante nommée de la forme :

```
com.sun.star.sdb.CommandType.TABLE
```

Le contenu de la propriété `Command` dépend de ce choix, ainsi que le montre le tableau 17-6.

Tableau 17–6 Relations entre Command et CommandType

CommandType	Agit sur	Contenu de Command
TABLE	Une table de la source de données.	Le nom de la table.
QUERY	Une requête enregistrée.	Le nom de la requête enregistrée.
COMMAND	Une requête SQL.	Une chaîne de caractères contenant la requête.

Dans cet exemple, choisissons la constante TABLE et indiquons dans la propriété Command le nom de la table client. En fait, notre RowSet va déduire de ce choix l'instruction SQL implicite :

```
SELECT * FROM client
```

Nous voyons donc que les trois possibilités reviennent à demander l'exécution d'une instruction SQL. Ce sera fait en appelant la méthode execute. À titre de démonstration, nous affichons le nom du premier client obtenu dans le ResultSet intégré au RowSet. Pour finir, nous détruisons la ressource RowSet avec sa méthode dispose.

La seconde méthode pour connecter un RowSet à une source de données consiste à le « greffer » sur une connexion existante, en utilisant la propriété activeConnection. Ainsi, si une connexion est définie par ailleurs dans votre code, elle peut être réutilisée, ce qui permet d'économiser les ressources du système. Il est tout à fait possible de greffer plusieurs RowSet sur la même connexion.

```
rem Code17-02.sxw  bibli : avecRowSet Module2
Option Explicit

Sub CreerRowSetGreffe()
Dim unRowSet As Object

ConnecterSource
unRowSet = createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  .activeConnection = maConnexion
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = "client"
  .execute ' effectuer la requête implicite
  .afterLast ' aller au-delà du dernier enregistrement
  if .IsRowCountFinal then
    MsgBox("Nombre d'enregistrements : " & .RowCount)
  else
    MsgBox("Erreur, fin de table pas atteinte !", 16)
  end if
  .dispose ' détruire le RowSet
End With
DeconnecterSource
End Sub
```

Nous reprenons notre routine ConnecterSource, qui nous initialise une connexion dans la variable maConnexion. Il suffit d'affecter cette variable à la propriété activeConnection pour que le RowSet profite de ses caractéristiques.

Nous profitons de l'exemple pour vous montrer une autre caractéristique du RowSet : la propriété RowCount indique le nombre de lignes lues. La propriété IsRowCountFinal vaut

`True` si toutes les lignes ont été lues, et donc si `RowCount` indique le nombre total de lignes. Ici, ce n'est pas très utile, mais si, par la suite, vous insérez et/ou supprimez de multiples enregistrements, `RowCount` sera mis à jour.

Tri supplémentaire

Vous pouvez effectuer un tri sur le résultat de la commande. Le `ResultSet` obtenu en tiendra compte. On utilise pour cela la propriété `Order`, de type `String`. Elle contient les arguments habituellement mis après un `ORDER BY` du langage SQL.

```
rem Code17-02.sxw  bibli : avecRowSet Module3
Option Explicit

Sub TrierResultat()
Dim unRowSet As Object

ConnecterSource
unRowSet = createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  .activeConnection = maConnexion
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = "produits"
  .Order = "LIBELLE DESC"
  .execute ' effectuer la requête implicite
  Do While .next
    print "Libellé = " & .Columns.getByName("LIBELLE").String
  Loop
  .dispose ' détruire le RowSet
End With
DeconnecterSource
End Sub
```

Cette macro produit un `ResultSet` avec le contenu de la table `produits`, mais trié par ordre alphabétique décroissant sur les libellés de produits. Un tri en ordre croissant s'écrirait :

```
  .Order = "LIBELLE"
```

ou encore :

```
  .Order = "LIBELLE ASC"
```

Plusieurs critères de tris simultanés sont acceptés. Voici comment trier la table `client` :

```
  .Order = "NOM, PRENOM, NAISS DESC"
```

Les enregistrements sont classés par ordre alphabétique des noms, ordre alphabétique des prénoms en cas de même nom, et ordre inverse des années de naissance en cas de mêmes nom et prénom.

Filtre supplémentaire

La propriété `Filter` du `RowSet` permet de préciser quels enregistrements sont recherchés parmi ceux du résultat brut. Elle contient les arguments habituellement mis après un `WHERE` du langage SQL. Il est nécessaire d'affecter la valeur `True` à la propriété `ApplyFilter`.

```
rem Code17-02.sxw  bibli : avecRowSet Module4
Option Explicit

Sub FiltrerResultat()
Dim unRowSet As Object

ConnecterSource
unRowSet = createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  .activeConnection = maConnexion
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = "produits"
  .Filter = "PRIX < 3.9"
  .ApplyFilter = True
  .execute ' effectuer la requête implicite
  Do While .next
    print "Libellé = " & .Columns.getByName("LIBELLE").String
  Loop
  .dispose ' détruire le RowSet
End With
DeconnecterSource
End Sub
```

Avec notre pilote, le filtre n'accepte qu'un seul critère. Pour une valeur numérique, les opérateurs de comparaison habituels (les mêmes que pour Basic) sont utilisables.

Pour un champ texte, outre ces opérateurs qui s'appliquent sur l'ordre alphabétique, nous disposons aussi d'une recherche générique. Le filtre suivant recherche tous les libellés commençant par « Dent » :

```
.Filter = "LIBELLE LIKE 'Dent%'"
```

Celui-ci recherche les libellés *ne commençant pas* par « Dent » :

```
.Filter = "LIBELLE NOT LIKE 'Dent%'"
```

D'autres possibilités existent, consultez un manuel de langage SQL.

Utiliser les requêtes pré-enregistrées

Nous avons vu que `CommandType` permet d'associer une requête pré-enregistrée à un `RowSet`. Nous utiliserons la requête enregistrée créée au début de ce chapitre.

```
rem Code17-02.sxw  bibli : avecRowSet Module5
Option Explicit

Sub UtiliserRequeteEnregistree()
Dim unRowSet As Object

ConnecterSource
unRowSet = createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  .activeConnection = maConnexion
  .CommandType = com.sun.star.sdb.CommandType.QUERY
  .Command = "Clients nés après 1950"
  .execute ' effectuer la requête enregistrée
  Do While .next
    print "Nom = " & .Columns(1).String & "     né en " & .Columns(3).Int
  Loop
  .dispose ' détruire le RowSet
End With
DeconnecterSource
End Sub
```

Les événements du RowSet

En complément de la manipulation des enregistrements, l'objet `RowSet` engendre des événements permettant un contrôle fin des actions. Le principe des gestionnaires d'événements, ou *Listeners*, est exposé au chapitre 19.

Ces événements peuvent être interceptés au moyen de `listeners` à deux niveaux :

- avant l'action, pour approbation (*Approve* en anglais), avec le service `com.sun.star.sdb.XRowSetApproveListener` ;
- après l'action grâce à `com.sun.star.sdbc.XRowSetListener`.

Ces deux `listeners` sont affectés au `RowSet` respectivement par les méthodes `addRowSetApproveListener` et `addRowSetListener`. L'exemple suivant est intéressant à exécuter pas à pas dans l'EDI, en cliquant sur le bouton « Étape de procédure ».

```
rem Code17-02.sxw  bibli : avecRowSet Module6
Option Explicit

Sub IntercepterEvenements()
Dim unRowSet As Object
Dim ecoute1 As Object, ecoute2 As Object
Dim leNom As String, lePrenom As String
```

```
ConnecterSource
unRowSet = createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  ' mise en place des écoutes
  ecoute1 = createUnoListener("ecouteAvant_", _
            "com.sun.star.sdb.XRowSetApproveListener")
  .addRowSetApproveListener(ecoute1)
  ecoute2 = createUnoListener("ecouteApres_", _
            "com.sun.star.sdbc.XRowSetListener")
  .addRowSetListener(ecoute2)

  ' ------  travail sur le RowSet ----

  .activeConnection = maConnexion
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = "client"
  .execute
  .next
  .next
  leNom = .Columns(1).String
  lePrenom = .Columns(2).String
  MsgBox("Modification de " & leNom & " " & lePrenom)
  .updateString(3, "Aristide") ' changer le prénom
  .updateRow
  .moveToInsertRow
  .updateInt(1, 50)
  .updateString(2, "Inconnu")
  .updateString(3, "Lola")
  .updateInt(4, 1985)
  .updateInt(5, 0)
  .insertRow ' ajouter cet enregistrement
  .execute ' recharger le RowSet
  .last
  ' supprimer les écoutes
  .removeRowSetApproveListener(ecoute1)
  .removeRowSetListener(ecoute2)
End With
DeconnecterSource
End Sub

'--- gestionnaires des événements XRowSetApproveListener ---

Function ecouteAvant_approveCursorMove(eve As Object) As Boolean
dim rep As Long
rep = Msgbox("Passer à une autre ligne ?",4)
ecouteAvant_approveCursorMove = (rep = 6)
End Function
```

```
Function ecouteAvant_approveRowChange(eve As Object) As Boolean
dim rep As Long
rep = MsgBox("Modifier la ligne ?", 4)
ecouteAvant_approveRowChange =  (rep = 6)
End Function

Function ecouteAvant_approveRowSetChange(eve As Object) As Boolean
MsgBox("Le RowSet va changer")
ecouteAvant_approveRowSetChange=true
End Function

Sub ecouteAvant_disposing()  ' nécessaire, même si pas utilisé
End Sub

'--- gestionnaires des évènements XRowSetListener ---

Sub ecouteApres_cursorMoved(eve As Object)
MsgBox "On est passé à une autre ligne"
End Sub

Sub ecouteApres_rowChanged(eve As Object)
MsgBox "La ligne vient d'être modifiée"
End Sub

Sub ecouteApres_rowSetChanged(eve As Object)
MsgBox "Le RowSet vient de changer"
End Sub

Sub ecouteApres_disposing()  ' nécessaire, même si pas utilisé
End Sub
```

Après avoir obtenu un objet `RowSet`, nous nous en servons pour mettre en place l'interception d'événements. Ici, nous traitons les deux jeux d'événements, mais nous pourrions nous limiter par exemple aux événements d'approbation. Le premier argument de `createUnoListener` est le préfixe qui sera utilisé pour chaque routine de gestion des événements concernés. Le préfixe doit se terminer par le caractère « souligné ».

Pour chaque service d'événements intercepté, chacun d'eux doit être traité par un sous-programme `Sub` ou `Function`, y compris l'événement `disposing` qui existe toujours, et pour lequel on ne fait souvent rien de particulier. Chaque routine d'interception reçoit en argument (optionnel) un objet événement. Les routines d'approbation sont des fonctions booléennes qui doivent renvoyer `True` pour approuver l'événement, `False` pour l'annuler.

Dans la séquence de travail avec le `RowSet`, nous avons mis en gras les instructions qui vont déclencher des événements (avant, après).

Le traitement des `listeners` est très important dans la validation des données à entrer ou pour contrôler l'exécution de votre code. Nous verrons au chapitre 18 que les formulaires donnent un accès simplifié à ces événements.

Les transactions

Tout au long de l'exposé précédent, nous avons pu constater que toute modification sur la base de données intervenait immédiatement. Il est cependant des cas où des traitements longs faisant intervenir plusieurs requêtes d'insertion ou de mises à jour sont à valider d'une seule traite. C'est la notion de transaction.

> **ATTENTION Limitation de dBase**
>
> Le format de base retenu pour nos exemples, dBase, ne permet pas les transactions. Pour exécuter nos exemples, vous devez donc utiliser une base de données adéquate, MySQL par exemple.

Ouverture de la base

Le principe pour créer la connexion est le même que précédemment. Cependant, la connexion que nous allons créer, afin de pouvoir gérer les transactions, ne doit pas être partagée avec d'autres composants. Nous faisons donc appel à la méthode `getIsolatedConnection`.

```
rem Code17-02.sxw  bibli : Transactions Module1
Option Explicit

Sub TransactionParDefaut()
Dim methode As String

ConnecterSourcePourTransaction
print "Validation immédiate ? " & maConnexion.autoCommit
Select Case maConnexion.TransactionIsolation
Case com.sun.star.sdbc.TransactionIsolation.NONE
  methode = "NONE"
Case com.sun.star.sdbc.TransactionIsolation.READ_UNCOMMITTED
  methode = "READ_UNCOMMITTED"
Case com.sun.star.sdbc.TransactionIsolation.READ_COMMITED
  methode = "READ_COMMITED"
Case com.sun.star.sdbc.TransactionIsolation.REAPEATABLE_READ
  methode = "REAPEATABLE_READ"
Case com.sun.star.sdbc.TransactionIsolation.SERIALIZABLE
  methode = "SERIALIZABLE"
End Select
print "Méthode utilisée : " & methode

DeconnecterSource
End Sub
```

```
Sub ConnecterSourcePourTransaction()
Dim NomSource As String, login As String, password As String
Dim maSource As Object, monDbContext As Object
'Création du contexte
NomSource = "BaseExemple"
monDbContext = CreateUnoService( _
                  "com.sun.star.sdb.DatabaseContext")
maSource=monDbContext.getByName(NomSource)

'Les paramètres de connexion
login = ""
password = ""
maConnexion = maSource.getIsolatedConnection(login, password)
End Sub
```

Le comportement par défaut des connexions est de valider les modifications dès l'instruction soumise par une instruction `execute`. C'est la propriété `autoCommit` qui indique ce comportement. Si nous désirons entrer dans un processus de transaction, ils nous faut donc modifier ce comportement par défaut (`True`) pour informer la connexion qu'elle doit attendre un ordre explicite avant d'inscrire les modifications en base.

Les écritures dans la base étant donc asynchrones, il se pose le problème de l'accès concurrent aux données. Il est possible d'indiquer à la connexion comment l'accès concurrent va être géré, si le moteur de la base de données le permet. C'est la propriété `IsolatedConnection` qui l'indique. Ces différentes méthodes sont regroupées dans les constantes nommées du tableau 17-7, qui sont de la forme :

```
com.sun.star.sdbc.TransactionIsolation.NONE
```

Tableau 17–7 Constantes de TransactionIsolation

Constante	Description
NONE	Les transactions ne sont pas prises en charge.
READ_UNCOMMITTED	Permet à une autre connexion de lire les changements avant qu'ils ne soient validés (`commit`). Si la transaction est annulée (`rollback`), cette autre connexion récupèrera un enregistrement invalide .
READ_COMMITED	Empêche la lecture d'un enregistrement modifié et non validé (`commit`) par une autre connexion.
REAPEATABLE_READ	Empêche le croisement des modifications entre deux transactions. Empêche qu'une première transaction lise une valeur, qu'une deuxième la modifie puis que la première relise cette valeur, obtenant alors une résultat incohérent.
SERIALIZABLE	Comme précédemment mais s'appliquant également aux clauses `WHERE`.

La propriété `TransactionIsolation` ne peut être modifiée que si le pilote autorise des niveaux d'isolation.

Gérer les transactions

Afin d'en illustrer l'utilisation, nous présentons l'exemple suivant demandant la confirmation lors de l'insertion d'un enregistrement. Les transactions s'appliquent bien entendu à des cas beaucoup plus complexes.

```
rem Code17-02.sxw  bibli : Transactions Module2
Option Explicit

' ceci ne fonctionne pas avec une table dBase !
Sub TestTransaction()
dim maRequete As Object
' maConnexion est déclarée public
dim temp As String, instrSQL As String, message As String
dim reponse As Long, nbLignesInscrites As Long

ConnecterSourcePourTransaction
AfficherNbreEnregistrements("client")

' contourner une bogue du pilote ODBC MsAccess
temp = maConnexion.dbg_properties
maConnexion.autoCommit = false ' on ouvre la transaction

'Requête d'insertion (ici tous les champs ne sont pas remplis)
instrSQL = "insert into client (NOM, PRENOM) " & _
          "values('César','Jules')"
maRequete = maConnexion.createStatement()
nbLignesInscrites = maRequete.executeUpdate(instrSQL)

message = "Nombre de lignes inscrites : " & nbLignesInscrites
message = message & chr(13) & chr(13) & "Confirmez-vous ?"
reponse = msgbox(message, 48+4)

if reponse=6 and nbLignesInscrites=1 then
  maConnexion.commit 'OUI
else
  maConnexion.rollback 'NON
endif

maConnexion.autoCommit = true ' fermer la transaction
AfficherNbreEnregistrements("client")
DeconnecterSource
End Sub

Sub AfficherNbreEnregistrements(nomTable As String)
' utilise la connexion déjà établie
Dim unRowSet As Object
```

```
unRowSet = createUnoService("com.sun.star.sdb.RowSet")
With unRowSet
  .activeConnection = maConnexion
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = nomTable
  .execute
  .afterLast ' aller au-delà du dernier enregistrement
  if .IsRowCountFinal then
    MsgBox("Nombre d'enregistrements : " & .RowCount)
  else
    MsgBox("Erreur, fin de table pas atteinte !", 16)
  end if
  .dispose
End With
End Sub
```

Nous ouvrons donc une connexion dans le mode nécessaire. Nous comptons le nombre d'enregistrements avant l'opération afin de pouvoir comparer l'influence des deux méthodes que nous allons voir. Le sous-programme `AfficherNbreEnregistrements` reprend un mécanisme déjà utilisé dans un exemple précédent.

La transaction est démarrée en désactivant le mode de validation automatique. Nous exécutons alors notre requête d'insertion tout à fait normalement. La requête est validée du côté de la base puisque le nombre d'enregistrements `nbLignesInscrites` retourné vaut 1.

La macro demande donc ensuite d'accepter ce changement. Si l'utilisateur répond oui alors la transaction est validée par la méthode `commit`. Si l'utilisateur répond non, toute la transaction est annulée par la méthode `rollback`.

Nous fermons alors la transaction en replaçant la connexion dans son mode par défaut de validation automatique.

La macro affiche alors le nombre d'enregistrements de la table, ce qui permet de vérifier qu'une action `commit` ajoute bien l'enregistrement et que `rollback` annule les changements.

Utilisation dans le contexte bureautique

OpenOffice.org n'est pas un gestionnaire de bases de données et l'intérêt de l'API sur les bases de données est de pouvoir utiliser ces fonctionnalités dans un contexte bureautique. Nous présentons maintenant quelques exemples non exhaustifs des traitements possibles.

Le publipostage

Une utilisation fréquente des sources de données en bureautique est le publipostage. Il permet par exemple de remplir une lettre-type avec des informations propres à chaque enregistrement de la source de données. À titre d'exemple, nous allons effectuer un publipostage sur la table `client` de notre source de tests `BaseExemple`.

La lettre type aura été préparée auparavant à l'aide de l'outil mailing du traitement de texte. Nous utiliserons le fichier `ExemplePublipostage.sxw`, fourni dans le zip téléchargeable.

```
rem Code17-03.sxw  bibli : Standard Module1
Option Explicit

Sub Publipostage()
Dim NomSource as string
Dim DocumentModele as string, RepertoireCible as string
Dim NomTable as string, NomChampTitreFichier as string
Dim monPublipostage As Object, MyProps()

'Paramètres  à adapter à votre configuration
NomSource = "BaseExemple"
DocumentModele = "C:\Docs OpenOffice\ExemplePublipostage.sxw"
RepertoireCible = "C:\Docs OpenOffice\resultat"
NomTable = "client"
NomChampTitreFichier = "NOM"

'Définition du publipostage
monPublipostage = createUnoService( _
                        "com.sun.star.text.MailMerge")
With monPublipostage
  .DataSourceName = NomSource
  .CommandType = com.sun.star.sdb.CommandType.TABLE
  .Command = NomTable
  .OutputType = com.sun.star.text.MailMergeType.FILE
  .FileNameFromColumn = True
  .FilenamePrefix = NomChampTitreFichier
  .DocumentURL = ConvertToURL(DocumentModele)
  .OutputURL = ConvertToURL(RepertoireCible)
  'Envoi de la commande
  .execute(MyProps())
End With
MsgBox "Fin du publipostage"
End Sub
```

La fonction de publipostage est directement accessible depuis le service `MailMerge`. Il donne accès à beaucoup des objets que nous avons évoqués jusqu'à maintenant en en simplifiant l'accès dans le cas de l'utilisation du publipostage. Nous commençons donc par créer un objet `monPublipostage` de ce type. Nous remplissons ensuite les diverses propriétés nécessaires au paramétrage du publipostage, voir le tableau 17-8.

Tableau 17–8 Propriétés de MailMerge

Propriété	Type	Signification
`DataSourceName`	`String`	Nom de la source de données.
`CommandType`	`Long`	Origine des données (voir `RowSet`).
`Command`	`String`	Nom de la table.
`OutputType`	`Long`	Type de sortie (fichier ou imprimante) - C'est une constante nommée pouvant prendre une des valeurs suivantes : `com.sun.star.text.MailMergeType.FILE` `com.sun.star.text.MailMergeType.PRINTER` `com.sun.star.text.MailMergeType.MAIL` (à partir de la version OOo 2.0)
`FileNameFromColumn`	`Boolean`	`True` si le nom du fichier de sortie dépend du contenu d'une colonne.
`FileNamePrefix`	`String`	Nom du champ engendrant le préfixe du fichier.
`DocumentURL`	`String`	URL du modèle du publipostage.
`OutputURL`	`String`	URL du répertoire de destination - Doit exister.

Comme nous le voyons, le publipostage s'appuie sur un `CommandType` que nous mettons à `TABLE`. Ceci ouvre la perspective de créer dynamiquement la source du publipostage en se basant sur une requête que l'on construirait dynamiquement suivant le contexte. Nous indiquons ensuite la table source `client` dans la propriété `Command`.

Nous choisissons une sortie sur fichier avec le paramètre `OutputType`. Avec la version 1.1.x d'OpenOffice.org, chaque enregistrement produira un fichier différent. Vous pourrez vous servir éventuellement de la macro `FusionneDocuments` disponible sur le site fr.openoffice.org pour les rassembler tous dans un document unique. Nous définissons dans ce cas les règles de nommage des fichiers créés. Avec la version 2.0 d'OpenOffice.org, la propriété booléenne `SaveAsSingleFile` sauvegarde le résultat dans un fichier unique.

La propriété booléenne `FileNameFromColumn` permet de construire un nom de fichier dépendant du contenu d'un champ du publipostage et le `FileNamePrefix` donne le nom de ce champ.

Nous indiquons le fichier de modèle du publipostage en renseignant l'URL `DocumentURL` ainsi que le répertoire contenant le résultat avec `OutputURL`. Ce répertoire doit exister.

Enfin, le publipostage est effectivement lancé en appelant la méthode `execute` de `monPublipostage`. L'argument tableau `Args()` de la méthode permet de gérer le synchronisme du processus. Nous ne traiterons pas cet aspect.

Une fois le message de fin affiché, le répertoire destination contient un fichier par enregistrement issu du publipostage.

À partir de la version 2.0 d'OpenOffice.org (et non la version 1.1.2 comme l'indique le SDK de la version 2.0), le paramètre `OutputType` peut prendre la valeur `MAIL`, pour créer des courriers électroniques. Vous disposez alors de toute une série de paramètres précisant la constitution des courriels. Nous vous renvoyons à leur description dans le SDK version 2.0, à la page du service `com.sun.star.text.MailMerge`.

Une requête dans Calc

Calc peut également être utilisé conjointement à une source de données. En tant que tableur, il est à même de manipuler des données, pour créer des graphes par exemple.

> OpenOffice.org permet d'utiliser un document Calc comme base de données, mais il est seulement possible de lire les données, non de les modifier ; cette section utilise Calc en tant que *tableur*, et non pas comme base de données.

Le fichier `CalcSQL.sxc` regroupe quelques fonctions permettant d'accéder aux sources de données depuis Calc. Sans donner le détail de tout le code, nous en mentionnerons les points importants.

Ces fonctions s'appuient sur le fait qu'une macro, définie sous forme de fonction, peut être appelée depuis une cellule de Calc. Bien que n'apparaissant pas dans l'autopilote, le résultat retourné par la fonction, sous forme matricielle ou non, sera affiché dans Calc au même titre qu'une fonction intrinsèque.

Une formule matricielle s'obtient en validant la saisie de la fonction dans Calc par Ctrl+Maj+Entrée à la place de la seule touche Entrée. Du coté de la macro, ce retour matriciel se traduit par un résultat sous forme de tableau.

Une requête dans une cellule

Une première approche est d'exploiter directement l'API d'accès aux sources de données que nous avons vue jusqu'à maintenant. Voici à quoi pourrait ressembler l'appel depuis une cellule Calc (la ligne est coupée en deux par la mise en page).

```
=CALCSQL1("SourceDeDonnees";"select * from LaTable
where LeChamp=" & QUOTE(A14))
```

On remarque que les arguments sont séparés par des points-virgules et que la concaténation de chaînes est autorisée par l'intermédiaire de l'opérateur `&`. N'omettez pas d'utiliser la fonction `QUOTE` si nécessaire. Cette syntaxe permet donc l'utilisation de valeurs issues d'autres cellules pour construire la requête, par exemple `A14`.

La fonction `CalcSQL1` contient plusieurs arguments, certains obligatoires, d'autres optionnels, voir tableau 17-9.

Tableau 17–9 Arguments de la fonction CalcSQL1

Ordre	Argument	Optionnel	Signification
1	NomSource	N	Nom de la source de données telle que définie dans OpenOffice.org - La casse est importante
2	Requete	N	Requête SQL valide de type SELECT. Aucun contrôle de validité autre que celui interne à OpenOffice.org n'est effectué.
3	AfficheTitre	O	La valeur 1 permet d'afficher le nom des champs récupérés.
4	BorneMax	O	Affiche les `BorneMax` premiers éléments.
5	BorneMin	O	Saute les `BorneMin` premiers éléments.

Ces arguments de la formule Calc sont en fait les arguments de la macro. Il est cependant à noter que les paramètres optionnels ne le sont que s'ils sont situés à la fin de la formule. Si ce n'est pas le cas, il deviennent obligatoires.

```
Function CalcSQL1(NomSource, Requete, _
            Optional AfficheTitre, _
            Optional BorneMax, _
            Optional BorneMin)
```

Les traitements de cette macro sont exactement dans l'esprit des macros que nous avons explorées précédemment. Nous ne re-détaillerons pas le processus. Une fois le traitement effectué, la macro retourne un tableau

```
CalcSQL1=TabResultat()
```

La cellule Calc contient alors le résultat de la requête.

REMARQUE Résultats matriciels

Il existe plusieurs problèmes dûs aux retours matriciels.

- Toutes les cellules affichées sont recalculées : cela implique des appels successifs aux fonctions qui peuvent être pénalisants si la requête est lourde.
- Lorsque la fonction a été évaluée une première fois, la taille de la matrice n'est pas adaptée si l'on change la requête. Cela implique l'affichage de #NA si moins de lignes sont retournées. Si plus de lignes sont retournées, le retour est tronqué à la taille initiale.

Importer des données

Le tableur Calc permet, par l'intermédiaire de la touche F4, d'afficher le pilote de sources de données. À partir de cet endroit, toute requête ou table glissée-déposée dans une feuille importe les données. Le lien n'est cependant pas conservé.

La fonction CalcSQL2 utilise cette API d'importation des données afin de reproduire et d'automatiser ce comportement. La requête source est alors conservée.

```
=CALCSQL2("Feuille";"Cellule";"Source";"select * from LaTable
where LeChamp=" & QUOTE(A14))
```

La fonction CalcSQL2 importe donc le résultat de la requête à l'endroit spécifié par les arguments. Le résultat est cellule par cellule et non pas matriciel. Les arguments de cette fonction (tableau 17-10) sont tous obligatoires.

Tableau 17–10 Arguments de la fonction CalcSQL1

Ordre	Argument	Signification
1	NomFeuille	Feuille où sera affiché le résultat - Doit être différente de la feuille contenant la cellule.
2	CelluleCible	Cellule en haut à gauche contenant le résultat.
3	NomSource	Nom de la source de données telle que définie dans Ooo - La casse est importante.
4	Requete	Requête SQL valide de type SELECT. Aucun contrôle de validité autre que celui interne à OpenOffice.org n'est effectué.

On remarque que les arguments sont séparés par des points-virgules et que la concaténation de chaînes est autorisée par l'intermédiaire de l'opérateur &. N'omettez pas d'utiliser la fonction QUOTE si nécessaire. Cette syntaxe permet donc l'utilisation de valeurs issues d'autres cellules pour construire la requête, par exemple A14.

L'objet plage de cellules de Calc expose une méthode d'importation de données que nous allons exploiter ici.

```
dim ParamSource(2) as new com.sun.star.beans.PropertyValue

FeuilleRes=thisComponent.sheets.getByName(NomFeuille)
LeCellRange=FeuilleRes.getCellRangeByName( _
                      CelluleCible & ":" & CelluleCible)

'Importation Base
paramSource(0).name="DatabaseName"
paramSource(0).value=NomSource'"SDDGrosseTable"
paramSource(1).name="SourceType"
paramSource(1).value=com.sun.star.sheet.DataImportMode.SQL
paramSource(2).name="SourceObject"
'exemple : "select * from grosseTable where numligne>12500"
paramSource(2).value=Requete

'Importe
LeCellRange.doImport(paramSource())
```

Nous commençons par définir la plage de cellules cibles `leCellRange`. Seule la colonne supérieure gauche est nécessaire. Si besoin est, le résultat débordera sur les cellules adjacentes. Les propriétés de l'importation sont ensuite renseignées. On peut remarquer une certaine analogie de ces propriétés avec ce qui est défini dans le tableau 17-6. L'esprit est effectivement identique. Le nom du service change dans ce cas : `com.sun.star.sheet.DatabaseImportDescriptor`.

La propriété `DatabaseName` indique le nom de la source de données telle que définie au sein du gestionnaire.

Les constantes nommées de la propriété `SourceType`, voir tableau 17-11, conditionnent le contenu de `SourceObject`. Elles sont de la forme :

```
com.sun.star.sheet.DataImportMode.NONE
```

Tableau 17–11 Valeurs possibles de SourceType

Constante	Signification
NONE	Aucune importation n'est effectuée.
SQL	La source de l'importation est une requête SQL.
TABLE	La source de l'importation est une table.
QUERY	La source de l'importation est une requête enregistrée.

Enfin, l'appel de la méthode `doImport` sur la plage de cellules effectue le traitement en lui-même. Le fichier `CalcSQL.sxc` propose une dernière fonction `CalcSQL3` comprenant plus d'options de mise en forme. Cette fonction reprend les deux approches exposées ci-dessus et nous ne nous y attarderons pas.

Importer les requêtes MS-Access

Nous avons vu qu'il était possible de créer dynamiquement des requêtes enregistrées. Cette technique va nous permettre d'importer les requêtes d'une base MS-Access. Ainsi, lors d'une migration de base de données, nous pourrons continuer à utiliser les requêtes, parfois complexes, qui y étaient enregistrées.

Deux macros, une en VBA pour l'exportation, l'autre en OOoBasic pour l'importation en passant par un fichier texte intermédiaire, vont permettre de recréer ces requêtes dans la source de données utilisée par OpenOffice.org.

```
Rem ATTENTION, ceci est du code VBA
Rem Il n'est pas utilisable dans OpenOffice.org

Rem Code VBA à executer dans Access
```

```
Sub ExporteRequetes()
Dim mabase As database
Set mabase = Currentdb
Set mesdefs = mabase.QueryDefs

Open "Result.txt" For Output As #1

For i = 0 To mesdefs.Count - 1
  Print #1, CStr(i) + "|------------------"
  Print #1, mesdefs(i).Name
  Print #1, mesdefs(i).Sql 'peut être écrit sur plusieurs lignes
  Print #1, ""
Next i
Print #1, "|||"
Close #1

MsgBox "ok"
End Sub
```

Cette première macro VBA parcourt toutes les requêtes de la base Access et les exporte dans un fichier texte.

Maintenant, la macro suivante va relire ce fichier dans OpenOffice.org, en tenant compte des éventuels sauts de lignes inclus dans les requêtes, et va les recréer dans la source de données.

```
Sub ImporteRequetes()
Dim rien as string

oDBContext= CreateUnoService("com.sun.star.sdb.DatabaseContext")
'Nom de la source de données OOo
odb=odbcontext.getbyname("EssaiMDB")
queries=odb.getQueryDefinitions()

Open "C:\Result.txt" For Input As #1

line input #1, rien
while not eof(1)
  NomRequete=""
  requete=""
  suite=""
  'Recupère les informations Info de chaque requête
  line input #1, NomRequete
  line input #1,requete
  line input #1,suite
  while trim(suite)<>""
    requete=requete+" "+suite
    line input #1,suite
  wend
  while trim(suite)=""
    line input #1, suite
  wend
```

```
  'Insère la requête dans la source de données
  UneRequete=queries.createInstance
  UneRequete.setPropertyValue("Command",requete)
  if not queries.hasByName(NomRequete) then
    'on ne peut écrire deux requêtes avec le même nom
    queries.InsertByName(NomRequete,UneRequete)
  endif
wend

on Error goto OOoV1
dbDoc = odb.DatabaseDocument 'erreur avant OOo 2.0 !
on Error Goto 0 ' supprimer le traitement d'erreur
dbDoc.store ' sauver la requête dans le fichier .odb
Goto L1
OOoV1:
Resume L1  ' igorer l'erreur : OOo 1.1 ou inférieur
L1:
on Error Goto 0

close #1
Msgbox "C'est Fini"
End Sub
```

Cette macro est un premier jet. En toute rigueur, il faudrait vérifier que les requêtes sont syntaxiquement correctes et ne contiennent pas d'éléments étrangers à la norme SQL. Nous vous laissons poursuivre ce travail...

Conclusion

Nous venons de présenter le traitement des sources de données au travers de l'API. Les différentes actions de création, modification ou consultation des données ont été exposées. Quelques exemples d'utilisation dans un contexte bureautique ont permis d'illustrer les concepts expliqués.

Le chapitre suivant va présenter la manipulation des formulaires directement inclus dans les documents et qu'on peut connecter à des sources de données.

18

Les formulaires

Nous traitons dans ce chapitre de certains aspects de programmation concernant les formulaires. Nous supposons que vous savez créer un formulaire et le connecter à une base de données. Si OpenOffice.org 2.0 permet de créer des formulaires intégrés à un document Base, il est toujours possible de définir un formulaire séparé, comme sur les versions précédentes. Nous allons décrire d'abord les formulaires séparés, puis nous montrerons comment accéder aux formulaires intégrés dans un document Base.

Dans la barre d'outils des contrôles de formulaires (voir figure 18-1), signalons l'existence d'un Navigateur de formulaires (figure 18-2). Il est très pratique pour visualiser la hiérarchie des objets dans les formulaires complexes, se positionner dans un formulaire, sélectionner un contrôle.



Figure 18–1
La barre d'outils de formulaire

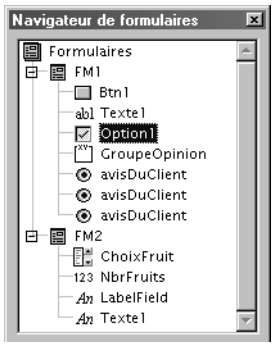


Figure 18–2
Le Navigateur de formulaires

Pour plus d'information, consultez l'aide en ligne, rubrique Fonctionnalité de la base de données, sous-rubrique Formes (mauvaise traduction de l'anglais *Forms*, qui signifie ici Formulaires).

Nous conseillons la lecture préalable du chapitre 17 traitant des sources de données, ainsi que du chapitre 15 sur les boîtes de dialogue ; leurs contrôles ont bien des points communs – mais aussi des différences – avec les formulaires.

Nous utiliserons dans les exemples la source de données `Bibliography` et la source `BaseExemple` que nous avons installées dans le chapitre 17. Pour les exemples spécifiques à la version 2.0 d'OpenOffice.org nous utiliserons la source représentée par le fichier `usine.odb` du chapitre 17.

Accéder aux contrôles d'un formulaire

Les formulaires sont déposés sur une page de dessin. Nous avons vu au chapitre 14 la notion de page de dessin dans les différents types de documents, lorsque nous avons décrit comment insérer des formes.

Pour accéder à un contrôle de formulaire, nous devons suivre un jeu de piste, résumé à la figure 18-3.

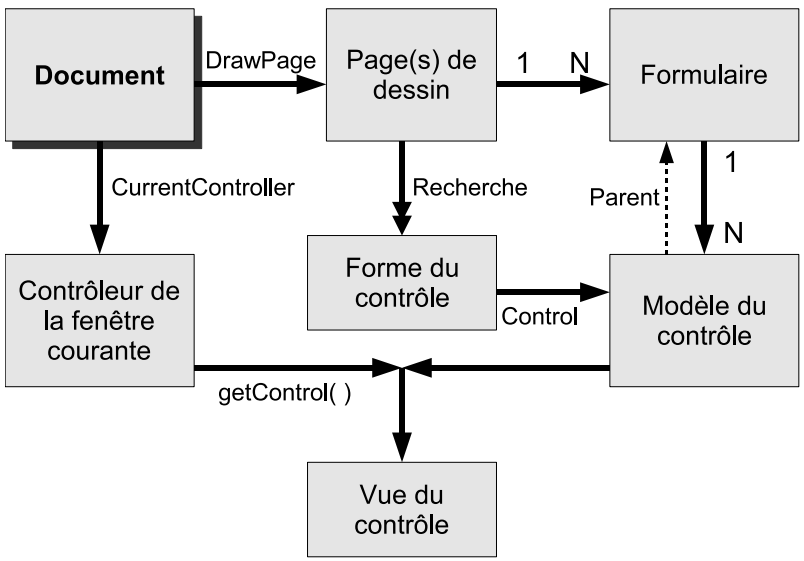


Figure 18–3
Comment retrouver un contrôle dans un formulaire

À partir du document, nous obtenons une page de dessin : soit celle du document (Writer), soit celle d'une de ses feuilles (Calc, Draw, Impress). Ceci a été décrit au chapitre 14.

Une page de dessin peut contenir un formulaire ou plusieurs. La propriété `Forms` de la page de dessin nous donne une collection de formulaires (collection éventuellement vide). Cette collection est capable de nous indiquer si un formulaire d'un nom donné existe, avec sa fonction `hasByName`, et de nous renvoyer ce formulaire avec la fonction `getByName`. Le formulaire est lui-même une collection, qui contient des contrôles. De la même manière, les fonctions `hasByName` et `getByName` d'un formulaire permettent de vérifier l'existence et d'obtenir le *modèle* d'un contrôle. Celui-ci contient l'essentiel des propriétés utiles du contrôle.

Contrairement aux boîtes de dialogue, le modèle est l'objet contrôle principal. Il existe en plus un objet « vue du contrôle » qui gère les propriétés visibles de celui-ci. Il n'y a pas de lien modèle vers vue, car il peut y avoir plusieurs vues s'il existe plusieurs fenêtres sur le même document. Toutefois, les vues restent cohérentes entre elles et avec le modèle. Pour obtenir la vue du contrôle, il est nécessaire de passer par le contrôleur de la fenêtre courante, qui est capable de renvoyer, avec sa fonction `getControl`, la vue du modèle de contrôle donné en argument.

> Comme le contrôleur de Draw ou Impress n'expose pas de fonction `getControl`, il est impossible de retrouver la vue du contrôle. Draw et Impress ne proposent pas totalement les fonctionnalités de formulaire.

Enfin, un contrôle est aussi une forme, puisqu'il est déposé sur une page de dessin. Cette forme nous donne accès, entre autres, à sa position et ses dimensions. Pour la retrouver, il est nécessaire d'examiner successivement les formes existant dans la page afin de récupérer celle qui correspond bien au contrôle. Pour cela, nous avons réalisé une routine utilitaire que nous détaillerons.

La macro ci-dessous met en œuvre le processus décrit. Vous remarquerez que notre algorithme se base sur les noms des diverses entités pour retrouver sûrement l'élément désiré. Il est de votre intérêt de choisir vous-même ces noms, plutôt que de laisser ceux donnés par défaut. Dans le document Calc utilisé par la macro, nous avons mis dans la feuille `Formulaires` deux formulaires, appelés `FM1` et `FM2`. Nous allons rechercher un contrôle appelé `Texte1` dans le formulaire `FM1`. Nous avons poussé le vice jusqu'à appeler `Texte1` un autre contrôle du formulaire `FM2`.

```
rem Code18-01.sxc  bibli : TrouverControles Module1
Option Explicit

Sub TrouverParLesNoms()
Dim monDocument As Object, maFeuille As Object, maPage As Object
Dim monControleur As Object
Dim lesFormulaires As Object, unFormulaire As Object
Dim monCtrl As Object, vueCtrl As Object, formeCtrl As Object
monDocument = thisComponent
monControleur = monDocument.CurrentController
```

```
maFeuille = monDocument.Sheets.getByName("Formulaires")
maPage = maFeuille.DrawPage ' récupérer la page de dessin
lesFormulaires = maPage.Forms ' la collection des formulaires
unFormulaire = lesFormulaires.getByName("FM1")
monCtrl = unFormulaire.getByName("Texte1")
monCtrl.Text = Time ' changer le texte affiché dans ce contrôle
vueCtrl = monControleur.GetControl(monCtrl)
print "Le contrôle Texte va disparaître"
vueCtrl.Visible = false
print "Le contrôle Texte va réapparaître"
vueCtrl.Visible = true
' rechercher la forme correspondant au contrôle
' attention : il existe aussi un controle Texte1 dans FM2 !
formeCtrl = FindCtrlShapeByName(maPage, "FM1", "Texte1")
print "Taille :", formeCtrl.Size.Width, formeCtrl.Size.Height
End Sub
```

Nous récupérons dans l'ordre la feuille de Calc, la page de dessin correspondant à la feuille, la collection de formulaires, puis le formulaire appelé `FM1` et, dans celui-ci, le (modèle de) contrôle appelé `Texte1`. La propriété `Text` de ce contrôle champ de texte correspond au texte affiché ; nous le modifions en lui affectant l'heure courante.

Une fois obtenue la vue du contrôle, nous démontrons que nous avons le bon objet en jouant avec sa propriété `Visible`. La forme (dessin) du contrôle est obtenue grâce à la fonction utilitaire `FindCtrlShapeByName`, que nous allons maintenant expliquer.

```
rem Code18-01.sxc   bibli : Standard Module1
Option Explicit

' retrouve la forme d'un controle de formulaire
Function FindCtrlShapeByName(unePage As Object, _
         leFormulaire As String, leControle As String) As Object
Dim objX As Object, x As Long

for x = 0 to unePage.Count -1
  objX = unePage(x)
  if objX.supportsService("com.sun.star.drawing.ControlShape") then
    if objX.Control.Name = leControle then
      if objX.Control.Parent.Name = leFormulaire then
        FindCtrlShapeByName = objX ' forme trouvée
        Exit Function
      end if
    end if
  end if
next
End Function ' renvoie Null en cas d'échec
```

Cette fonction va analyser chacune des formes se trouvant dans la page, en effectuant des tests successifs : d'abord, elle vérifie que la forme rend le service `ControlShape` (le nom du

service de formes pour contrôles). Si c'est le cas, sa propriété `Control` nous donne le contrôle auquel elle est liée ; nous vérifions que le nom de ce contrôle est bien celui recherché. Pour éviter les homonymies entre plusieurs formulaires, nous récupérons le formulaire dans lequel se trouve ce contrôle, grâce à sa propriété `Parent`. Si le nom du formulaire est celui attendu, l'objet forme est trouvé et renvoyé en résultat de la fonction. En cas d'échec, la fonction renverra la valeur `Null`, que l'appelant pourra tester avec la fonction `IsNull` de OOoBasic. Si, dans la macro précédente, vous modifiez dans l'appel de `FindCtrlShapeByName` le nom du formulaire en `FM2`, vous trouverez l'autre contrôle de même nom.

Les sous-formulaires

Un sous-formulaire n'est pas accessible directement à partir de l'ensemble des formulaires de la page. En fait, il est vu comme un contrôle appartenant au formulaire parent, et on l'obtient de la même façon qu'un autre contrôle :

```
Dim lesFormulaires As Object
Dim unFormulaire As Object, sousFormulaire As Object
unFormulaire = lesFormulaires.getByName("Sujets photos")
sousFormulaire = unFormulaire.getByName("Cliches")
```

Une fois en possession du sous-formulaire, vous avez accès de la même façon aux contrôles qu'il contient, et éventuellement à un second niveau de sous-formulaire, etc.

Fonctionnalités de base des contrôles

Un formulaire n'est pas obligatoirement lié à une base de données. Il peut servir à obtenir des renseignements qui seront ensuite imprimés, ou servir à déclencher des macros du document.

Les contrôles sont pour la plupart similaires à ceux rencontrés dans les boîtes de dialogue. D'une manière générale, le modèle d'un contrôle de formulaire est équivalent au modèle du contrôle analogue d'une boîte de dialogue et la vue d'un contrôle de formulaire est similaire au contrôle lui-même de la boîte de dialogue. Il existe cependant de notables différences.

Les propriétés de chaque contrôle sont nombreuses, tant au niveau du contrôle qu'au niveau de son modèle. Vous trouverez dans le zip téléchargeable, dans le répertoire consacré à ce chapitre, le fichier `PropsFormulaire.sxc` qui contient la liste des propriétés utiles de chacun des contrôles.

Le bouton

Un bouton de formulaire possède, comme son homologue de boîte de dialogue, une propriété Type de bouton, mais les valeurs possibles sont différentes :

- Push correspond à un bouton ordinaire. Pour réaliser une action, on doit gérer un événement de ce bouton.
- Reset réinitialise tous les champs du formulaire à leur valeur par défaut.
- Submit est utilisé pour envoyer le formulaire à un serveur Web.
- URL lance une adresse URL.

Le traitement d'un événement est identique à ce qui a été vu avec les boîtes de dialogue.

Le bouton picto, ou bouton-image

Ce contrôle est spécifique aux formulaires. Il sert à afficher une image, sans texte, et se déclenche comme un bouton. On retrouve la propriété Type de Bouton.

Les zones de liste

Lorsque vous déposez une zone de liste sur un formulaire, l'AutoPilote entre en action. Annulez-le pour imposer une liste de choix.

La zone de liste simple

Son fonctionnement est identique à celui d'une boîte de dialogue. Toutefois ici, il faut utiliser les propriétés et méthodes de la vue du contrôle. Dans la macro suivante, nous affichons le rang et la valeur choisie dans une zone de liste à choix unique, puis nous sélectionnons le choix de rang 1.

```
rem Code18-01.sxc  bibli : ManipControles Module2
Option Explicit

Sub ZoneDeListe()
Dim monDocument As Object, maFeuille As Object
Dim lesFormulaires As Object, unFormulaire As Object
Dim monCtrl As Object, vueCtrl As Object
monDocument = thisComponent
maFeuille = monDocument.Sheets.getByName("Formulaires")
lesFormulaires = maFeuille.DrawPage.Forms
unFormulaire = lesFormulaires.getByName("FM2")
monCtrl = unFormulaire.getByName("ChoixFruit")
vueCtrl = monDocument.CurrentController.GetControl(monCtrl)
print "Rang= " & vueCtrl.SelectedItemPos, vueCtrl.SelectedItem
print "Sélection du choix de rang 1"
vueCtrl.selectItemPos(1, True)
End Sub
```

Il n'y a rien de particulier à signaler pour une zone simple à choix multiple.

La zone de liste combinée

Son utilisation est identique à celle vue dans une boîte de dialogue.

La case à cocher

Son utilisation est identique à celle vue dans une boîte de dialogue. Ici aussi, la propriété `State` (État) du contrôle peut prendre une des valeurs :

0 La case n'est pas cochée.

1 La case est cochée.

2 L'état est « indéterminé » ou « je ne sais pas » si le statut triple est activé.

Le choix 1 parmi N

La manière de créer un ensemble de cases de choix 1 parmi N (ou boutons de radio) est très différente de ce que nous avons vu avec les boîtes de dialogue. C'est la parfaite illustration qu'il y a plusieurs solutions à un même problème...

La manière la plus simple de créer un tel ensemble est de déposer une zone de groupe. Un Autopilote vous guidera dans les étapes :

1. Donner l'intitulé de chaque bouton (le texte affiché).
2. Sélectionner un bouton par défaut.
3. Donner une valeur à chaque option (la valeur référentielle, qui peut être un texte).
4. Donner l'intitulé de la zone de groupe (le texte affiché en haut du cadre).

Vous obtenez un ensemble de boutons régulièrement espacés et entourés par un cadre. Cliquez n'importe où dans le cadre : l'ensemble est sélectionné, vous pouvez le déplacer en faisant glisser la souris. Cliquez ailleurs pour désélectionner. Maintenant, faites un Ctrl + clic sur un des éléments ; celui-ci seulement sera sélectionné.

Dans un choix 1 parmi N de formulaire, tous les boutons reçoivent le même nom. C'est ainsi que le formulaire saura qu'ils fonctionnent ensemble. Donc, si vous souhaitez ajouter une autre option plus tard, déposez un contrôle `Bouton radio` et donnez lui ce même nom. Comment alors pourrons-nous distinguer le bouton choisi ? Nous avons besoin d'un élément spécifique. L'intitulé du bouton pourrait convenir, mais il est susceptible d'être modifié plus tard, soit pour le traduire dans une autre langue, soit pour des raisons de clarté. La valeur référentielle, définie à l'étape 3 convient mieux. Cependant, cette propriété `RefValue` du modèle est normalement prévue pour être envoyée à un serveur Web après validation du formulaire. Une autre solution totalement indépendante de l'usage du formulaire est d'utiliser la propriété `Tag` (Complément d'information) de chaque bouton. C'est ce que nous avons fait dans le document exemple.

La récupération du choix effectué par l'utilisateur est encore complètement différente de ce que nous avons vu pour une boîte de dialogue, comme le montre la macro suivante.

```
rem Code18-01.sxc  bibli : ManipControles Module1
Option Explicit

Sub Groupe1ParmiN()
Dim monDocument As Object, maFeuille As Object
Dim lesFormulaires As Object, unFormulaire As Object
Dim opinion() As Object, x As Long
monDocument = thisComponent
maFeuille = monDocument.Sheets.getByName("Formulaires")
lesFormulaires = maFeuille.DrawPage.Forms
unFormulaire = lesFormulaires.getByName("FM1")
unFormulaire.getGroupByName("avisDuClient", opinion())
for x = 0 to UBound(opinion())
  With opinion(x)
    MsgBox("Index= " & x & "  Label= " & .Label & chr(13) & _
           "    Tag=  " & .Tag & "    Etat= " & .State)
  End With
next
End Sub
```

Après avoir récupéré l'objet formulaire, nous utilisons sa méthode `getGroupByName`. Le premier argument est le nom commun à tous les boutons du groupe (attention, ne pas confondre avec le nom du contrôle zone de groupe). La méthode affectera au deuxième argument un tableau d'objets, chacun d'eux étant un des boutons. Signalons qu'il n'y a aucune raison de supposer que les éléments du tableau sont dans l'ordre des boutons, car les boutons ont pu être déplacés en mode conception. Le tableau obtenu est exploré ; pour la démonstration, nous affichons l'index du tableau, l'intitulé du bouton. La valeur de la propriété `Tag` sert d'élément de reconnaissance. La propriété `State` vaut 1 si le bouton est choisi, 0 s'il ne l'est pas.

Naturellement, il est aussi possible d'utiliser un événement comme « Lors du déclenchement » sur chacun des boutons.

Les champs de saisie

Il s'agit des contrôles :

- zone de texte,
- champ numérique,
- champ date,
- champ horaire,
- champ monétaire,
- champ masqué,
- champ formaté.

Leur utilisation est identique à celle vue dans une boîte de dialogue.

Particularités :

- Champ date - Si la valeur par défaut n'existe pas, la date courante est affichée.
- Champ heure - Si la valeur par défaut n'existe pas, l'heure courante est affichée.

Autres contrôles

Il s'agit des contrôles :

- sélection de fichier,
- image (picto).

Leur utilisation est identique à celle vue dans une boîte de dialogue.

Contrôles et base de données

Certains contrôles peuvent être liés à une base de données, ce qui permet à l'utilisateur de visualiser le contenu de certains éléments et de les modifier facilement. Il est important de se rappeler les points suivants :

- Un formulaire est lié au maximum à une source de données (une base de données).
- Dans cette base de données, un formulaire ne peut gérer qu'une seule table, ou résultat d'une instruction SQL. Le résultat d'une instruction SQL est l'équivalent d'une table créée dynamiquement.
- Les contrôles orientés données affichent un champ dans cette table (sauf le contrôle `Table`, qui en affiche plusieurs).
- Les sous-formulaires permettent de gérer d'autres tables de la même base de données que le formulaire principal auquel ils sont rattachés.

L'objet formulaire

Le formulaire permet d'accéder à une table et d'y naviguer. Il possède les propriétés et méthodes que nous avons vues au chapitre 17 pour une table et pour un `RowSet`.

Le contrôle Table

Le contrôle `Table`, ou plutôt grille (anglais `Grid`) est le moyen permettant à l'utilisateur de se déplacer dans une table, de visualiser ses champs, de changer leur contenu, d'insérer ou de supprimer des enregistrements. Il est donc essentiel dans un formulaire de base de données.

Chaque colonne d'un contrôle Table est lui-même un des contrôles que nous avons déjà vus. L'objet Table expose ses colonnes sous forme d'une collection dont chaque élément est accessible par indexation ou par le nom (propriété Name) de la colonne. Ne pas confondre avec le libellé de la colonne, propriété Label.

Dans l'exemple suivant, nous avons créé un formulaire sur la table client de la base de données du chapitre 17. Dans ce formulaire, un contrôle Table a été déposé en utilisant l'AutoPilote, pour afficher les colonnes NOM et PRENOM. Puis, nous avons ajouté la colonne numérique SITUATION. Nous avons laissé les valeurs par défaut des propriétés Name des colonnes.

```
rem Code18-02.sxw  bibli : ModifTable1 Module1
Option Explicit

Sub ExplorerFormulaire()
Dim monDocument As Object, unFormulaire As Object
Dim grille As Object, uneColonne As Object
Dim x As Long

monDocument = thisComponent
unFormulaire = monDocument.DrawPage.Forms.getByName("MaJClient")
grille = unFormulaire.getByName("TableControl")
for x = 0 to grille.Count -1
  uneColonne = grille(x)
  MsgBox ("Rang de la colonne = " & x & chr(13) & _
    "Nom = " & uneColonne.Name & chr(13) & _
    "Titre = " & uneColonne.Label & chr(13) & _
    "Type du contrôle = " & uneColonne.ColumnServiceName)
next
uneColonne = grille.getByName("Col1")
MsgBox("Valeur du champ SITFAM = " & uneColonne.Value)
End Sub
```

Nous commençons par récupérer dans le document le formulaire nommé MaJClient, puis le contrôle Table nommé TableControl. La simple indexation de cet objet permet d'accéder aux colonnes, dont le nombre total se trouve dans la propriété Count. La boucle For affiche successivement, pour chaque colonne, le nom de celle-ci (nous avons gardé les valeurs données automatiquement), le libellé qui est visible dans l'en-tête de colonne, et le type du contrôle ce qui nous permet de constater que les deux premières colonnes sont des champs Texte (contrôle Edit), et la troisième un champ numérique.

Nous récupérons ensuite directement la colonne correspondant au champ SITFAM dans la table. Remarquez comme le nom utilisé n'a rien à voir, ni avec la signification de la colonne, ni même avec son rang. Conclusion : pensez à renommer le nom des colonnes d'un contrôle Table. Le contenu affiché est celui du champ correspondant dans l'enregistrement en cours.

Le contenu d'un champ d'une colonne s'obtient en employant la propriété donnant la valeur pour le type de contrôle utilisé pour la colonne : le contenu du champ NOM s'obtiendrait avec la propriété Text de la colonne correspondante. Pour la colonne SITFAM, qui est un contrôle numérique, on utilisera la propriété Value.

Connaître la sélection dans le contrôle Table

L'utilisateur dispose de plusieurs moyens de sélection : placer le curseur dans une des cellules affichées par le contrôle Table ; ou bien sélectionner une ligne en cliquant sur la marge gauche du contrôle ; ou encore sélectionner plusieurs lignes avec Maj+clic et/ou Ctrl+clic sur la marge ; enfin, sélectionner le contenu entier en cliquant sur la marge en haut à gauche. Le codage ci-dessous traite ces différents cas.

```
rem Code18-02.sxw  bibli : ModifTable1 Module3
Option Explicit

Sub SelectionDansControleTable()
Dim monDocument As Object, unFormulaire As Object
Dim grille As Object, vueGrille As Object
Dim lignesSel As Variant, rs As Object
Dim x As Long
monDocument = thisComponent
unFormulaire = monDocument.DrawPage.Forms.getByName("MaJClient")
rs = unFormulaire.createResultSet
grille = unFormulaire.getByName("TableControl")
vueGrille = monDocument.CurrentController.getControl(grille)
lignesSel = vueGrille.Selection
if UBound(lignesSel) < 0 then
  MsgBox("Pas de ligne sélectionnée. Position courante :" & chr(13) & _
     grille(1).Text & " " & grille(0).Text & " : " & _
     grille.getByName("Col situ").Value & chr(13) & _
     "Le curseur est en colonne " & vueGrille.CurrentColumnPosition & _
     " du contrôle Table")
else
  for x = 0 to UBound(lignesSel)
   if rs.moveToBookmark(lignesSel(x)) then
     ' accès par rang de colonne du ResultSet, à partir de 1
     MsgBox(rs.getString(3) & " " & rs.getString(2) & " : " & _
            rs.getDouble(5) )
   else
     MsgBox("Signet non trouvé ! " & lignesSel(x), 16)
   end if
  next
end if
End Sub
```

L'analyse se fait à partir de la vue du contrôle, obtenue via la propriété CurrentController du document. Dans cet objet vueGrille, la propriété Selection est un

tableau représentant les lignes sélectionnées. Si le tableau est vide (son index maximal est négatif), il y a au plus une cellule sélectionnée, ou le curseur est dans une cellule.

Dans ce premier cas, le modèle du contrôle `Table` nous fournit les valeurs de tous les champs de la ligne, en l'indexant par le numéro de colonne, débutant à zéro. Une méthode alternative consiste à retrouver la colonne à partir de son nom interne, qu'on trouve en mode conception en cliquant sur l'en-tête de colonne puis en affichant les propriétés du contrôle. Dans notre exemple, la colonne `SITUATION` a pour nom interne `Col situ`. Enfin, la propriété `CurrentColumnPosition` de la vue du contrôle nous renseigne sur la position de la cellule courante, relativement aux colonnes du controle `Table`.

Dans le cas où le tableau n'est pas vide, il contient un signet (`bookmark`) par ligne sélectionnée. L'utilisation des signets a été expliquée au chapitre 17, section « Exploiter les résultats d'une requête ». Nous avons auparavant créé un `ResultSet` basé sur le formulaire. Il nous permet d'accéder au bon enregistrement dans la table sous-jacente. Attention : ici nous affichons les valeurs de l'enregistrement avec des index de colonne relatifs au `ResultSet`, donc sans rapport avec la disposition des colonnes dans le contrôle `Table`.

Déplacer la position courante dans le contrôle Table

En fait, déplacer la ligne courante du contrôle `Table` est très simple. Il suffit d'utiliser les méthodes de navigation du `ResultSet` du formulaire, comme nous l'avons fait au chapitre 17.

```
rem Code18-02.sxw  bibli : ModifTable1 Module4
Option Explicit

Sub SeDeplacerDansLeControleTable()
Dim monDocument As Object, unFormulaire As Object

monDocument = thisComponent
unFormulaire = monDocument.DrawPage.Forms.getByName("MaJClient")
MsgBox("Position actuelle : " & unFormulaire.Row & chr(13) & _
        "Cliquez OK pour aller à la ligne 2")
unFormulaire.absolute(2)
MsgBox("Position actuelle : " & unFormulaire.Row & chr(13) & _
       "Cliquez OK pour aller à la dernière ligne")
unFormulaire.last
MsgBox("Position actuelle : " & unFormulaire.Row & chr(13) & _
       "Cliquez OK pour remonter de 4 lignes")
unFormulaire.relative(-4)
MsgBox("Position actuelle : " & unFormulaire.Row & chr(13) & _
       "Cliquez OK pour aller à la ligne suivante")
unFormulaire.next
MsgBox("Position actuelle: " & unFormulaire.Row)
End Sub
```

Les contrôles champ de données

Les contrôles Edit, Date, Heure, Nombre, Monétaire, Formaté, Masqué sont orientés données et peuvent être liés à une colonne de la table du formulaire. À cet effet, ils disposent de propriétés spécifiques, listées au tableau 18-1, et d'une méthode spécifique, `commit`.

Tableau 18–1 Propriétés de contrôle orienté données

Propriété	Type	Signification
`DataField`	`String`	Nom de la colonne qu'affiche le contrôle.
`BoundField`	`Object`	L'objet colonne effectivement lié au contrôle, ou `Null`.
`DataFieldProperty`	`String`	Nom de la propriété du contrôle qui contient la valeur.

Le document exemple disponible dans le zip téléchargeable contient trois contrôles orientés données. L'un affiche un champ texte de la table, un autre est volontairement connecté à un champ qui n'existe pas, le troisième affiche un champ numérique de la table. Nous allons montrer l'utilisation des propriétés et comment se servir d'un contrôle pour modifier le contenu de la table.

```
rem Code18-02.sxw  bibli : ModifTable1 Module2
Option Explicit

Sub LiensControleTable()
Dim monDocument As Object, unFormulaire As Object
Dim unCtrl As Object
monDocument = thisComponent
unFormulaire = monDocument.DrawPage.Forms.getByName("MaJClient")

unCtrl = unFormulaire.getByName("leNom")
afficherControle(unCtrl) ' contrôle champ texte

unCtrl = unFormulaire.getByName("SituFamille")
afficherControle(unCtrl) 'contrôle numérique mal connecté

unCtrl = unFormulaire.getByName("DateNaiss")
afficherControle(unCtrl) ' contrôle champ numérique

unCtrl.Value = InputBox("Année de naissance ?", , unCtrl.Value)
unCtrl.commit ' transférer la valeur dans la base de données
End Sub

Sub afficherControle( k As Object)
Dim valeur As Variant, cr As String, ConnexOK As Boolean
cr = chr(13) ' retour à la ligne
valeur = k.getPropertyValue(k.DataFieldProperty)
ConnexOK = not IsNull(k.BoundField)
```

```
MsgBox("Nom du contrôle = " & k.Name & cr & _
       "Champ de la table = " & k.DataField & cr & _
       "Contrôle connecté ? " & ConnexOK  & cr & _
       "Nom de la propriété de valeur = " & _
       k.DataFieldProperty & cr &  "Valeur = " & valeur)
End Sub
```

Le sous-programme `afficherControle` prend pour argument un objet contrôle (plus exactement, son modèle). Nous l'utiliserons sur chacun des contrôles orientés données du formulaire.

La propriété `DataFieldproperty` nous permet de récupérer facilement la valeur réelle dans la colonne liée, pour la ligne en cours de la table. La fonction `getPropertyValue`, disponible sur les objets API, renvoie la valeur de la propriété dont le nom exact est donné en argument. Ce nom est une chaîne de caractères dont la casse doit être respectée. Ne connaissant pas a priori le type de la valeur obtenue, nous utilisons une variable de type `Variant`.

La propriété `BoundField` fournit l'objet colonne auquel est lié le contrôle. Cependant, si pour une raison quelconque (erreur, table inaccessible...) le contrôle ne peut accéder à la colonne, cette propriété vaut `Null`, alors que la propriété `DataField` n'indique que le nom de la colonne à utiliser, même si elle est inaccessible. L'exécution du sous-programme `afficherControle` sur le contrôle mal connecté vous montre l'utilité de `BoundField`.

Nous modifions ensuite la valeur de la table à partir du contrôle `DateNaiss`. Comme nous savons qu'il s'agit d'un contrôle numérique, nous utilisons directement sa propriété `Value`, en l'affichant et en modifiant sa valeur selon la réponse de l'utilisateur. Cependant, ceci ne suffit pas : pour que cette valeur soit reportée dans la table, on doit utiliser la méthode `commit` du contrôle. Ici, la modification est artificielle et sans utilité pratique, mais le principe serait identique pour imposer une valeur particulière, par exemple tenant compte d'autres colonnes dans la ligne. Remarquez enfin que, si l'utilisateur donne une réponse quelconque ou clique sur Annuler, la valeur du champ est alors imposée par les limites haute et basse du contrôle.

BOGUE Modification non prise en compte

Modifiez la valeur de la date de naissance. Puis, avec la souris, cliquez sur la barre de navigation du contrôle Table pour passer à l'enregistrement suivant ou précédent. Revenez ensuite sur l'enregistrement initial : il n'a pas pris en compte la nouvelle valeur (vu sur la version 1.1.1).

Cette bogue ne se produit pas avec la barre de navigation d'enregistrement en bas de la fenêtre du document. Elle ne se produit pas non plus si vous changez le *focus* en appuyant sur la touche Tab.

Zone de liste et base de données

Les contrôles Zone de liste peuvent offrir un choix obtenu à partir d'une table. L'AutoPilote facilite la configuration du contrôle pour cet usage.

Dans la base de données exemple, la table `SitFam` associe un libellé à chacune des valeurs numériques possibles du champ `SITFAM` de la table `client`. Le document `Code18-03.sxw` du zip téléchargeable présente une zone de liste simple qui affiche les libellés de situation familiale. Le libellé affiché correspond à la valeur du champ `SITFAM` dans la ligne courante. Si vous choisissez un autre libellé, cela modifiera le contenu de la table client.

> **BOGUE Liste de choix**
>
> En cliquant sur la liste déroulante du contrôle zone de liste, vous observerez (version 1.1.1) une entrée vide.

La propriété `ListSourceType` de l'objet zone de liste (son modèle) correspond à Type de contenu de la liste dans l'onglet Données du panneau de propriétés du contrôle. Elle contient une constante nommée de la forme :

```
com.sun.star.form.ListSourceType.SQL
```

Les différentes valeurs possibles sont listées au tableau 18-2. Suivant la valeur de `ListSourceType`, la propriété `ListSource`, qui est un tableau de `String`, est utilisée différemment. Dans le cas SQL, le seul élément du tableau contient l'instruction SQL. Pour plus d'informations, consultez la documentation API sur le service `DatabaseListBox`, à la page :

```
com.sun.star.form.component.DatabaseListBox
```

Tableau 18–2 Constantes de ListSourceType

Constante	Remplissage de la liste
VALUELIST	Avec une simple liste de libellés
TABLE	Avec le contenu d'une table.
QUERY	Avec le résultat d'une requête.
SQL	Avec le résultat d'une instruction SQL.
SQLPASSTHROUGH	Avec le résultat d'une instruction SQL non évaluée par le moteur de bases de données.
TABLEFIELDS	Avec les noms de champs d'une table.

Les événements des formulaires

L'onglet Événements du panneau de propriétés de formulaire présente la plupart des événements utiles, ce qui peut éviter de mettre en place un `Listener` (le principe d'un gestionnaire d'événements, ou `Listener`, est exposé au chapitre 19).

Le tableau 18-3 liste les événements spécifiques aux formulaires de données, le terme anglais équivalent, la correspondance probable avec l'interface et la routine de `Listener` concernées, et l'indication si le gestionnaire a un droit de *veto* (V) sur l'événement. Pour de tels événements, la routine de gestion est une fonction qui renvoie `True` si l'événement est autorisé, `False` s'il doit être interdit. S'il y a plusieurs gestionnaires sur le même événement, la priorité est donnée à `False`.

Tableau 18–3 Événements sur un formulaire

Français	Anglais	Interface	Routine	V
Avant la restauration	Prior to reset	XResetListener	approveReset	V
Après la restauration	After resetting	XResetListener	resetted	
Avant l'envoi	Before submitting	XSubmitListener	approveSubmit	V
Lors du chargement	When loading	XLoadListener	loaded	
Avant le rechargement	Before reloading	XLoadListener	reloading	
Lors du rechargement	When reloading	XLoadListener	reloaded	
Avant le déchargement	Before unloading	XLoadListener	unloading	
Lors du déchargement	When unloading	XLoadListener	unloaded	
Confirmation de suppression	Confirm deletion	XConfirmDeleteListener	confirmDelete	V
Avant l'opération d'enregistrement	Before record action	XRowSetApproveListener	approveRowchange approveRowSetchange	V V
Après l'opération d'enregistrement	After record action	XRowSetListener	rowChanged rowSetChanged	
Avant le changement d'enregistrement	Before record change	XRowSetApproveListener	ApproveCursorMove	V
Après le changement d'enregistrement	After record change	XRowSetListener	cursorMoved	
Remplir les paramètres	Fill parameters	XDatabaseParameterListener	approveParameter	V
Erreur survenue	Error occurred	XSQLErrorListener	errorOccured	

REMARQUE XSQLErrorListener

Le nom de la routine `errorOccured` comporte une faute d'orthographe, qui sera perpétuée pour des raisons de compatibilité…

Le tableau 18-4 liste les événements concernant la base de données qui apparaissent sur l'onglet Propriétés des contrôles orientés données.

Tableau 18–4 Événements sur un contrôle

Français	Anglais	Interface	Routine	V
Avant la restauration	Prior to reset	XResetListener	approveReset	V
Après la restauration	After resetting	XResetListener	resetted	
Avant l'actualisation	Before updating	XUpdateListener	ApproveUpdate	V
Après l'actualisation	After updating	XUpdateListener	updated	
Erreur survenue	Error occurred	XSQLErrorListener	errorOccured	

Le document `Code18-03.sxw` démontre l'utilisation d'événements sur le formulaire. Il utilise plusieurs macros.

```
rem Code18-03.sxw  bibli : ModifTable2 Module1
Option Explicit

Global VoirEvenements As Boolean

Sub TracerLesEvenements()
VoirEvenements = not VoirEvenements
End Sub

' traitement d'événement sur le formulaire
' renvoie True pour accepter, False pour refuser le changement
Function AvantOperation(evt As Object) As Boolean
afficheInfos(evt, "Avant opération d'enregistrement")
if VoirEvenements then
  AvantOperation = _
     (MsgBox("Approuver cette modification ?", 4) = 6)
else
  AvantOperation = true
end if
End Function

' traitement d'événement sur le formulaire
' renvoie True pour accepter, False pour refuser le changement
Function AvantChangmtEnregistrement(evt As Object) As Boolean
afficheInfos(evt, "Avant changement d'enregistrement")
AvantChangmtEnregistrement = True
End Function

Sub afficheInfos(evt As Object, nomSub As String)
Dim nomEvt As String, posGuillemet As Long
if not VoirEvenements then exit sub
nomEvt = evt.Dbg_Methods
posGuillemet = InStr(nomEvt, """")
```

```
nomEvt = Mid(nomEvt, posGuillemet +1)
posGuillemet = InStr(nomEvt, """")
nomEvt = Left(nomEvt, posGuillemet -1)
MsgBox("Routine=" & nomSub & chr(13) & _
       "Evènement=" & nomEvt & chr(13) & _
       "Source=" & evt.Source.ImplementationName )
End Sub
```

Nous avons utilisé une variable globale afin de mémoriser son état entre deux lancements de macro. Initialement, cette variable a la valeur `False`, ce qui conduit les gestionnaires d'événements à ne rien faire. Exécutez une fois la macro `TracerLesEvenements`. La variable `VoirEvenements` prendra la valeur `True`.

Dans l'onglet Événements du panneau Propriétés du formulaire, nous avons intercepté les événements :

- Avant l'opération d'enregistrement,
- Avant le changement d'enregistrement.

Ces deux événements peuvent être rejetés (la documentation dit peu élégamment « vetoisé »). Nous les traitons donc par deux fonctions à résultat de type `Boolean`. Les deux gestionnaires commencent par appeler un sous-programme chargé d'afficher quelques informations sur l'événement.

La routine `afficheInfos` utilise la propriété `Dbg_Methods` pour récupérer (dans sa première ligne) le nom de l'événement. La propriété `Source` de l'objet Événement nous fournit l'objet l'ayant déclenché ; nous affichons le nom d'implémentation de l'objet, qui nous suffira pour le reconnaître.

La fonction `AvantChangmtEnregistrement` est déclenchée quand l'utilisateur déplace le pointeur sur un autre enregistrement. Ici, nous acceptons tout changement et renvoyons une valeur `True`.

La fonction `AvantOperation` est déclenchée lorsqu'un enregistrement modifié va être recopié dans la base de données. Ici, nous laissons le choix à l'utilisateur d'accepter ou non le changement. S'il répond Non, le résultat sera `False`. Un véritable programme pourrait effectuer un contrôle de cohérence entre champs avant de valider le changement, par exemple refuser un client veuf âgé de 10 ans...

Déplacer le curseur d'un enregistrement à l'autre déclenche bien l'événement attendu.

Modifiez un des champs dans un enregistrement, puis changez d'enregistrement. Ceci déclenche un événement Avant l'opération et l'affichage de la demande d'approbation. Répondez Oui. Un deuxième événement Avant l'opération est déclenché. En comparant les informations affichées, vous constaterez que la source est différente. Répondez Non à cette deuxième demande de validation. Le curseur reste sur le même enregistrement, le contrôle Table indique que l'enregistrement est en cours de modification (un crayon sur la

marge de gauche). Changez encore le curseur et répondez Oui aux deux demandes. L'événement Avant changement d'enregistrement apparaît, ce qui est normal.

En conclusion, la gestion des événements est assez délicate et le codage doit tenir compte des différents cas possibles. Exécutez une nouvelle fois la macro `TracerLesEvenements` pour ne plus voir les messages sur les événements.

Formulaire intelligent

Vous trouverez dans le SDK un exemple très instructif et réaliste d'un formulaire pour passer une commande à un « fast-food » : `burger_factory.odt` (pour le SDK version 1.1, il existe un document `sxw` équivalent). À chaque valeur introduite dans un contrôle, des *listeners* mettent à jour d'autres contrôles du formulaire. Vous trouverez le formulaire dans le sous-répertoire du SDK :

```
/examples/basic/forms_and_controls/
```

Principes communs aux contrôles

Imposer le focus sur un contrôle

Nous avons défini la notion de focus au chapitre 15 « Les boîtes de dialogue », et montré au chapitre 16 comment mettre le focus sur un contrôle de dialogue. La méthode est similaire pour un contrôle de formulaire. On utilise la méthode `setFocus` de la vue du contrôle concerné.

```
unCtrl = unFormulaire.getByName("DateNaiss")
vueCtrl = monDocument.CurrentController.getControl(unCtrl)
vueCtrl.setFocus
```

Gestionnaire d'événement commun à plusieurs contrôles

De même que pour les boîtes de dialogue, il est tout à fait possible d'affecter le même gestionnaire d'événement à plusieurs contrôles. Voici un exemple simple, que vous pouvez tester en mettant plusieurs boutons sur un document Writer, et en affectant un même gestionnaire à l'événement « Lors du déclenchement ». Pour chaque bouton, avec le panneau Propriétés de l'EDI, remplissez la propriété Complément d'information avec un texte quelconque, spécifique à chacun.

```
Sub GestionnaireMultiple(Evenement As Object)
Msgbox(Evenement.Source.Model.Tag)
End Sub
```

Notre gestionnaire récupère la vue du contrôle déclencheur avec la propriété `Source` de l'objet événement. Le modèle du contrôle est obtenu par la propriété `Model` de la vue, et la propriété `Tag` correspond à `Complément d'information` dans l'EDI. Vous pouvez aussi bien accéder aux autres propriétés, comme le nom du contrôle.

Les formulaires intégrés dans un document Base

Nous appellerons ici « formulaire intégré » un formulaire mémorisé dans un document `.odb` réalisé avec la version 2.0 d'OpenOffice.org. Dans cette version, les formulaires intégrés sont basés sur des sous-documents de type Writer. Il existe un sous-document par formulaire intégré.

Un formulaire au sens de l'utilisateur d'un document Base comporte un formulaire principal et éventuellement des sous-formulaires, au sens API.

Ouvrir un formulaire d'un document Base

Nous allons ouvrir un formulaire intégré au fichier `usine.odb` (disponible dans le zip téléchargeable, répertoire du chapitre 17), et laisser l'utilisateur naviguer dans les données. Ceci est réalisé avec une première macro :

```
rem Code18-04.odt  bibli : Standard Module1
Option Explicit

Dim astuce As Object ' pour maintenir le formulaire affiché

Global maConnexion as Object, dbDoc As Object

Sub AfficherFormulaire()
Dim monDbContext As Object, maSource As Object
Dim monFormulaire As Object, lesFormulaires As Object
Dim OdbOpt(0) as new com.sun.star.beans.PropertyValue
Dim stdOpenArgs(1) as new com.sun.star.beans.PropertyValue
Dim adrOdb As String
Const nomSource = "usine", nomFormulaire = "Achats des Clients"

monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource = monDbContext.getByName(NomSource)
dbDoc = maSource.DatabaseDocument
adrOdb = dbDoc.URL
```

```
OdbOpt(0).Name = "Hidden"
OdbOpt(0).Value = True
dbDoc = StarDesktop.loadComponentFromURL(adrOdb, "_blank", 0, OdbOpt())
lesFormulaires = dbDoc.FormDocuments

' contournement de bogue : l'instruction suivante est
' indispensable pour visualiser le formulaire en fin de macro
astuce = lesFormulaires.getByName(nomFormulaire)

stdOpenArgs(0).Name = "ActiveConnection"
maConnexion = dbDoc.Datasource.getConnection("","")
stdOpenArgs(0).Value = maConnexion
stdOpenArgs(1).Name = "OpenMode"
stdOpenArgs(1).Value = "open"
lesFormulaires.loadComponentFromURL(nomFormulaire, _
                                    "_blank", 0, stdOpenArgs())
End Sub
```

D'abord, nous devons charger le document `.odb` qui contient (entre autres) les formulaires intégrés. Pour cela, il nous faut son adresse URL, que nous obtenons à partir de la propriété `DatabaseDocument` exposée par la source de données. Ici, nous avons choisi de charger le document sans l'afficher. Le conteneur des formulaires est récupéré par la propriété `FormDocuments`.

Maintenant, nous allons contourner une bogue de la version OpenOffice.org que nous utilisons pour ce livre. Sans précautions, le formulaire apparaît quelques dixièmes de secondes seulement. Cette partie est susceptible de changer en fonction des futures corrections d'OpenOffice.org. La variable `astuce` nous permet de garder le formulaire ouvert à la fin de la macro.

Le formulaire est ouvert avec la méthode `loadComponentFromURL` de l'objet collection de formulaires, en utilisant deux propriétés :

- `ActiveConnection` pour indiquer la connexion à utiliser ;
- `OpenMode` pour demander l'ouverture en mode actif, avec la valeur `open`.

La macro se termine ici, afin que l'utilisateur puisse utiliser le formulaire. Pour le refermer, et surtout refermer le document Base et la connexion, il faut maintenant lancer une deuxième macro :

```
Sub fermerFormulaire()
dbDoc.close(True)
maConnexion.close
maConnexion.dispose
End Sub
```

Les informations nécessaires à la fermeture ont été mémorisées par la première macro dans des variables de portée `Global` afin de les récupérer avec la deuxième macro.

Ouvrir un formulaire en mode conception

Si nous souhaitons seulement ouvrir un formulaire en mode conception, il n'est pas nécessaire d'établir une connexion avec la base. En ouvrant le document Base de manière visible, on laisse à l'utilisateur le soin de fermer formulaire et document Base. Le codage en est bien simplifié.

```
rem Code18-04.odt  bibli : Standard Module2
Option Explicit

Sub AfficherFormulaireConception()
Dim monDbContext As Object, maSource As Object, dbDoc2 As Object
Dim monFormulaire As Object, lesFormulaires As Object
Dim OdbOpt() as new com.sun.star.beans.PropertyValue
Dim stdOpenArgs(0) as new com.sun.star.beans.PropertyValue
Dim adrOdb As String
Const nomSource = "usine", nomFormulaire = "Achats des Clients"

monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource = monDbContext.getByName(NomSource)
dbDoc2 = maSource.DatabaseDocument
adrOdb = dbDoc2.URL
dbDoc2 = StarDesktop.loadComponentFromURL(adrOdb, "_blank", 0, OdbOpt())
lesFormulaires = dbDoc2.FormDocuments

' contournement de bogue : l'instruction suivante est
' indispensable pour visualiser le formulaire en fin de macro
lesFormulaires.getByName(nomFormulaire)

stdOpenArgs(0).Name = "OpenMode"
stdOpenArgs(0).Value = "openDesign"
lesFormulaires.loadComponentFromURL(nomFormulaire, _
                                    "_blank", 0, stdOpenArgs())
End Sub
```

Ici, il n'est pas nécessaire d'établir une connexion à la base de données. La propriété de chargement `OpenMode` doit prendre la valeur `openDesign`.

Obtenir les contrôles d'un formulaire intégré

Le processus est plus complexe qu'avec un document formulaire séparé, puisqu'il nous faut dans un premier temps récupérer le sous-document correspondant, puis trouver le formulaire au sens API, puis les contrôles. Nous allons lister ici les noms des contrôles de chaque formulaire intégré dans le document Base. Chaque sous-document formulaire sera affiché en mode conception, aussi la macro reprend-elle une partie de l'exemple précédent.

```
rem Code18-04.odt  bibli : Standard Module3
Option Explicit

Sub ListerControlesFormulairesIntegres()
Dim monDbContext As Object, maSource As Object, dbDoc2 As Object
Dim monFormulaire As Object, lesFormulaires As Object
Dim OdbOpt(0) as new com.sun.star.beans.PropertyValue
Dim stdOpenArgs(0) as new com.sun.star.beans.PropertyValue
Dim adrOdb As String, nomFormulaire As String
Dim f As Long, k As Long, x As Long, unCtrl As Object
Dim docFormulaire As Object, lesFormesDeDocForm As Object
Dim unFormulaire As Object
Const nomSource = "usine"

monDbContext = CreateUnoService("com.sun.star.sdb.DatabaseContext")
maSource = monDbContext.getByName(NomSource)
dbDoc2 = maSource.DatabaseDocument
adrOdb = dbDoc2.URL
OdbOpt(0).Name = "Hidden"
OdbOpt(0).Value = True
dbDoc2 = StarDesktop.loadComponentFromURL(adrOdb, "_blank", 0, OdbOpt())

stdOpenArgs(0).Name = "OpenMode"
stdOpenArgs(0).Value = "openDesign"

lesFormulaires = dbDoc2.FormDocuments
for f= 0 to lesFormulaires.Count -1
  monFormulaire = lesFormulaires(f)
  nomFormulaire = monFormulaire.Name
  ' contournement de bogue : l'instruction suivante est
  ' indispensable pour visualiser le formulaire
  lesFormulaires.getByName(nomFormulaire)
  docFormulaire = lesFormulaires.loadComponentFromURL(nomFormulaire, _
                                        "_blank", 0, stdOpenArgs())
  ' ici nous supposons que le document support est Writer
  lesFormesDeDocForm =  docFormulaire.DrawPage.Forms
  for x = 0 to lesFormesDeDocForm.Count -1
    unFormulaire = lesFormesDeDocForm(x)
    MsgBox("Formulaire du document .odb : " & nomFormulaire & _
      chr(13) & "Formulaire dans docFormulaire : " & unFormulaire.Name)
    for k = 0 to unFormulaire.Count -1
      unCtrl = unFormulaire(k)
      MsgBox("Contrôle " & unCtrl.Name)
    next
  next
next
dbDoc2.close(True)
End Sub
```

Après avoir chargé le document Base, nous explorons chacun de ses formulaires. Nous les récupérons grâce à leur nom « utilisateur » et la méthode `loadComponentFromURL` de la collection de formulaires.

Nous récupérons ensuite l'objet collection des formulaires API contenus dans le sous-document. Nous avons prévu une boucle, mais vous verrez qu'il n'y a qu'un seul formulaire API principal. Le message affiché pourra vous surprendre : le nom du formulaire API est différent du nom du formulaire du document Base. Il ne reste plus qu'à explorer les contrôles du formulaire API. Dans notre exemple, il existe un sous-formulaire qui est vu comme un contrôle. Les contrôles de ce sous-formulaire sont accessibles avec un codage supplémentaire, qui devrait être récursif pour traiter des cascades de sous-formulaires...

Conclusion

Les formulaires de document sont dotés d'outils spécifiques aux bases de données, dont nous avons exposé le traitement par les macros et l'API. La version 2.0 d'OpenOffice.org apporte les formulaires intégrés, plus pratiques pour l'utilisateur, mais plus complexes pour le programmeur. Ce chapitre clôt le traitement des sources de données au travers de l'API. Nous exposerons au chapitre suivant une partie de l'API dans un contexte hors document bureautique, illustrant ainsi la richesse et la puissance applicative qui sont à la disposition du développeur par le biais des macros.

19

Techniques avancées pour le poste de travail

Ce chapitre présente des manipulations et des macros utilitaires faisant appel à des aspects avancés de la programmation OpenOffice.org. Ces méthodes peu connues trouveront leur utilité dans la plupart des projets d'applications de grande envergure utilisant l'API d'OpenOffice.org comme un véritable environnement de développement dans lequel de nombreux outils sont disponibles.

Les répertoires d'installation

Disponible sous différents systèmes d'exploitation, OpenOffice.org permet à l'utilisateur de choisir le nom du répertoire principal d'installation. Pour éviter de coder « en dur » les adresses liées à l'installation ou au système d'exploitation, il est nécessaire d'utiliser les services `PathSettings` ou `PathSubstitution`.

Le service `PathSettings` connaît toutes les adresses de répertoires utilisés par la version d'OpenOffice.org qui exécute la macro. Son usage est tout simple : d'abord obtenir un accès au service, puis utiliser ses propriétés, qui sont toutes des chaînes de caractères listant un ou plusieurs répertoire(s).

Exemple :

```
Dim install As Object
install = CreateUnoService("com.sun.star.util.PathSettings")
MsgBox(install.Basic) ' répertoires des macros Basic
```

Les propriétés disponibles sont les suivantes :

```
Addin,  AutoCorrect, AutoText, Backup, Basic
Bitmap, Config, Dictionary, Favorite, Filter
Gallery, Graphic, Help, Linguistic, Module,
Palette, Plugin, Storage, Template,
UIConfig, UserDictionary,
Temp, Work
```

S'il existe plusieurs répertoires pour une propriété, ils sont séparés par un point-virgule. Ces chemins sont pour la plupart affichés dans le menu Outils > Options > OpenOffice.org > Chemins.

Les propriétés `Temp` et `Work` ne concernent pas l'installation, mais désignent respectivement le répertoire des fichiers temporaires et le répertoire principal de l'utilisateur.

Le service `PathSubstitution` offre d'autres facilités. Il remplace un mot-clé dans une chaîne de caractères par la valeur correspondante, en principe un chemin. La chaîne de caractères doit être compatible avec un format URL, car le résultat sera une URL. Ainsi, cette macro écrit un fichier de travail dans le répertoire dédié aux fichiers temporaires.

```
rem Code19-01.sxw   bibli : Chemins Module2
Option Explicit

Sub SubstituerChemin()
Dim fichTempo As String
Dim f1 As Integer, ps As Object

ps = CreateUnoService("com.sun.star.util.PathSubstitution")
fichTempo = "$(temp)/monFich$12345x.html"
fichTempo = ps.substituteVariables(fichTempo, false)
f1 = FreeFile
' écrire le fichier en écrasant un éventuel fichier existant
Open fichTempo for Output As f1
print #f1, "Bonjour tout le monde!"
Close #f1
End Sub
```

Le mot-clé `$(temp)` représente le répertoire utilisé pour les fichiers temporaires. La fonction `substituteVariables` du service va remplacer tous les mots-clés existant dans la chaîne de caractères en premier argument. Lorsque le deuxième paramètre vaut `true`, une erreur sera déclenchée en cas d'impossibilité de substituer. La fonction renvoie la chaîne

de caractères résultant de la substitution. Nous utilisons l'adresse obtenue pour créer le fichier et le remplir.

Le tableau 19-1 liste les variables de substitution disponibles. Le contenu d'une variable est fourni par la fonction `getSubstituteVariableValue` :

```
rem Code19-01.sxw   bibli : Chemins Module3
Option Explicit

Sub TestGetSubstituteVariableValue()
Dim ps As Object, s1 As String
ps = CreateUnoService("com.sun.star.util.PathSubstitution")
s1 = ps.getSubstituteVariableValue("$(vlang)")
MsgBox(s1)
End Sub
```

Tableau 19–1 Variables de substitution

Variable	Signification
`$(inst)`	Chemin d'installation d'OpenOffice.org.
`$(prog)`	Chemin du programme OpenOffice.org.
`$(user)`	Chemin du répertoire `user` dans l'installation.
`$(work)`	Chemin du répertoire de travail de l'utilisateur.
`$(home)`	Chemin du répertoire de l'utilisateur.
`$(temp)`	Chemin du répertoire des fichiers temporaires.
`$(path)`	Contenu de la variable d'environnement PATH ; les répertoires sont séparés par un point-virgule.
`$(lang)`	Numéro de pays de l'installation (France=33, USA=01).
`$(langid)`	Numéro de code de langue de l'installation (France=1036).
`$(vlang)`	Langage utilisé, en anglais (français=french).

Le chapitre 6.2.7 du « Developer's Guide » du SDK donne d'autres détails d'utilisation des services `PathSettings` et `PathSubstitution`.

Gérer les fichiers depuis l'API

Le service `SimpleFileAccess` offre des fonctions de gestion de fichiers (effacer un fichier, le copier, etc.) qui sont redondantes avec celles de Basic. Néanmoins, il apporte des possibilités complémentaires en ce qui concerne les fichiers binaires.

Écrire un fichier binaire

Nous allons créer un fichier, octet par octet. N'importe quelles valeurs peuvent être utilisées, mais pour faciliter la vérification, notre fichier ne contiendra que des octets ayant valeur de caractères.

```
rem Code19-01.sxw   bibli : FichBin Module1
Option Explicit

Public Const nomFichier = "C:\Docs OpenOffice\essai002.bin"

Sub EcrireFichierBinaire()
Dim f1 As Object, st As Object
Dim unPaquet(4) As Integer ' paquet de cinq octets

' il faut effacer tout fichier de ce nom
' autrement il sera simplement modifié, pas écrasé
if fileExists(nomFichier) then kill(nomFichier)
' constitution d'un paquet d'octets à écrire
unPaquet(0) = extSigne(78)   ' N
unPaquet(1) = extSigne(111)  ' o
unPaquet(2) = extSigne(235)  ' ë
unPaquet(3) = extSigne(108)  ' l
unPaquet(4) = extSigne(33)   ' !
f1 = createUnoService("com.sun.star.ucb.SimpleFileAccess")
st = f1.openFileWrite(convertToURL(nomFichier))
st.writeBytes(unPaquet())
st.flush()
st.closeOutput()
End Sub

' étend le bit signe de l'octet sur tout l'entier Integer
Function extSigne( unOctet As Integer) As Integer
if unOctet < 128 then
  extSigne = unOctet
else ' étend le bit signe (valeur 1) sur 8 bits à gauche
  extSigne = -256 or unOctet ' attention : OU binaire
end if
End Function
```

Adaptez le chemin du fichier à votre configuration ; la constante `nomFichier` est publique car nous l'utiliserons dans plusieurs modules.

Nous supprimons tout fichier existant, sinon le service `SimpleFileAccess` écrasera dans le fichier existant les octets que nous allons écrire (cela peut être intéressant dans certains cas).

Pour cet exemple, nous nous contenterons d'écrire cinq octets. Il est nécessaire de les fournir dans un tableau d'octets, mais Basic n'offrant pas ce type, nous devons utiliser des `Integer` (16 bits). Autre problème, le service attend des octets signés, c'est-à-dire entre -128 et +127,

alors qu'on a l'habitude de manipuler des octets non signés (valeur entre 0 et 255). Notre fonction `extSigne` réalise le transcodage en étendant vers la gauche le bit signe de l'octet.

On accède au service par `CreateUnoService`. La fonction `openFileWrite` de ce service nous renvoie un objet de « flux » (en anglais *stream*) qui nous permettra d'écrire le fichier. L'argument est l'URL du fichier. La méthode `writeBytes` de l'objet flux écrit les octets du tableau, dans l'ordre croissant de l'index. Dans un véritable programme, cette méthode serait utilisée autant de fois que nécessaire, en employant un tableau dont la taille peut varier selon les besoins.

La méthode `flush` sert à garantir que le tampon a bien été écrit physiquement avant de fermer le fichier par la méthode `closeOutput`.

Comme nous avons en fait écrit des caractères, vous pouvez vérifier le résultat avec un simple éditeur de texte.

Lire un fichier binaire

Nous allons relire par macro, et octet par octet, le fichier écrit précédemment.

```
rem Code19-01.sxw   bibli : FichBin Module2
Option Explicit

Sub LireFichierBinaire()
Dim f1 As Object, st As Object
Dim unPaquet() As Integer
Dim x As Long, nbOct As Long

f1 = createUnoService("com.sun.star.ucb.SimpleFileAccess")
st = f1.openFileRead(convertToURL(nomFichier))
nbOct = st.readBytes(unPaquet(), 100)
for x = 0 to nbOct-1
  print chr(unPaquet(x) and 255)
next
nbOct = st.readBytes(unPaquet(), 100)
print "Reste : " & nbOct
st.closeInput ' fermer le Stream
End Sub
```

La fonction `readBytes` renvoie le nombre d'octets lus et remplit le tableau en premier argument. Le deuxième argument est le nombre maximum d'octets que l'on souhaite lire. La lecture du fichier écrit précédemment donnera un nombre d'octets lus égal à 5, car la fin du fichier est atteinte au-delà. La deuxième instruction `readBytes` confirme que tout le fichier a été lu, car elle renverra une valeur nulle.

Comme l'octet lu est signé, il est nécessaire de le transformer en valeur positive, avec un ET logique avec la valeur 255 (hexadécimal 00FF).

Signalons l'existence de la méthode `skipBytes` de l'objet `stream`, qui sert à avancer dans le fichier du nombre d'octets indiqué en argument. Ceci est utile pour sauter des zones sans intérêt pour le travail envisagé.

```
st.skipByte(200) ' sauter les 200 octets suivants
```

Lecture-écriture d'un fichier binaire

La méthode `openFileReadWrite` ouvre un fichier en lecture et écriture. Il peut alors être lu avec la méthode `readBytes` ou la méthode `skipBytes`, puis écrit à partir de la dernière position de lecture.

Création et décompression d'un fichier ZIP

Le service `Package` de l'API est capable de lire ou écrire un fichier ZIP. Le fichier ZIP créé contient un fichier `manifest.xml` stocké dans le répertoire `meta-inf`, comme pour les documents OpenOffice.org (Writer, Calc, etc.). Quelques informations assez générales sont disponibles dans la documentation de l'API, à partir de la page `com.sun.star.packages`.

Le zip téléchargeable contient diverses macros pouvant servir d'outils de manipulation de ZIP. Nous ne les décrirons pas dans le détail, nous contentant de montrer leur utilisation.

```
rem Code19-01.sxw   bibli : Zipper Module1
Option Explicit

Private Const nomZip = "C:\Docs OpenOffice\monZipe.zip"
Private Const Fichier1 = "C:\Docs OpenOffice\ooo-1.1-cd-label.pdf"
Private Const Fichier2 = "C:\Docs OpenOffice\baratin.txt"

Sub EcrireZip()
Dim URLzip As String
URLzip = convertToURL(nomZip)
' créer un nouveau ZIP avec un fichier dedans
zipFile(URLzip, convertToURL(Fichier1), "")
' ajouter un fichier dans le répertoire textes/ de ce ZIP
zipFile(URLzip, convertToURL(Fichier2), "textes/")
print "Modifiez maintenant le fichier original Fichier2"
' mettre à jour le ZIP
zipFile(URLzip, convertToURL(Fichier2), "textes/")
End Sub
Sub supprimerZip()
Dim URLzip As String
URLzip = convertToURL(nomZip)
removeFileinZip(URLzip, "baratin.txt", "textes/")
End Sub
```

```
Sub listerContenuZip()
Dim URLzip As String, nomRep
URLzip = convertToURL(nomZip)
listZipContents(URLzip, "")
nomRep = InputBox("Répertoire du zip ? (xxx/)"
listZipContents(URLzip, nomRep)
End Sub

Sub dezipperFichier()
Dim URLzip As String
Const Fichier3 = "C:\Docs OpenOffice\Tatata.txt"
URLzip = convertToURL(nomZip)
UnzipAFile(URLzip, "textes/baratin.txt", convertToURL(Fichier3))
End Sub

Sub zipperArborescence()
Dim leZip As String, arbreAzipper As String
leZip = convertToURL("C:\Docs OpenOffice\essai1.zip")
arbreAzipper = convertToURL("C:\monArbre\")
ZipTree(lezip, arbreAzipper)
End Sub

Sub dezipperArchive()
Dim leZip As String, arbreDezip As String
leZip = convertToURL("C:\Docs OpenOffice\GrosWriter.sxw")
arbreDezip = convertToURL("C:\Dezip\")
unzipArchive(leZip, arbreDezip)
End Sub
```

On crée un fichier ZIP en donnant en arguments de la macro `zipFile` :

1. l'adresse URL du futur ZIP,
2. l'adresse URL d'un fichier à insérer dans le ZIP,
3. une chaîne de caractères vide,
4. éventuellement, le type MIME du fichier à insérer dans le ZIP.

Une fois le ZIP créé, la même macro sert à ajouter des fichiers, un par un. Si le fichier existe déjà dans le ZIP, il sera remplacé. Il est possible de stocker un fichier dans un répertoire à l'intérieur du ZIP ; dans ce cas, le troisième argument doit être le chemin de ce répertoire, terminé par le caractère `/`. Le type MIME, optionnel, est une chaîne de caractères, par exemple :

```
"text/xml" "image/png" "image/jpeg" "image/gif"
```

La macro `removeFileinZip` supprime un fichier contenu dans un ZIP. Ses trois arguments sont :

1. l'adresse URL du fichier ZIP,
2. le nom du fichier à supprimer du ZIP,

3 le chemin du répertoire de ZIP contenant le fichier à supprimer.

La macro `listZipContents` affiche les fichiers et répertoires contenus dans un répertoire d'un ZIP. Elle a pour arguments :

1 l'adresse URL du fichier ZIP,
2 le chemin d'un répertoire de ZIP (chaîne vide pour le répertoire racine).

La macro `UnzipAFile` décompresse un fichier d'un ZIP pour obtenir un fichier ordinaire. Elle utilise trois arguments :

1 l'adresse URL du fichier ZIP,
2 l'adresse URL du fichier dans le ZIP,
3 l'adresse URL du fichier final.

La macro `ZipTree` compresse tout un répertoire, avec ses fichiers et sous-répertoires. Toutefois, elle ne remplit pas le type MIME. Cette macro utilise deux arguments :

1 l'adresse URL du fichier ZIP,
2 l'adresse URL du répertoire à compresser (terminé par un `/`).

La macro `unzipArchive` décompresse une archive ZIP, avec ses fichiers et sous-répertoires, dans un répertoire de destination. Elle utilise deux arguments :

1 l'adresse URL du fichier ZIP,
2 l'adresse URL du répertoire de destination (terminé par un `/`).

Dans l'exemple, nous avons supposé la décompression d'un fichier Writer. Notez que la macro ne détecte pas les fichiers `mimetype` et `manifest.xml` qui existent dans les fichiers OpenOffice.org.

Lancer l'application associée à un document

Le service `SystemShellExecute` est capable, par exemple, d'afficher un document HTML sur le navigateur configuré par défaut sur le PC. Ceci nécessite toutefois une configuration correcte d'OpenOffice.org et du système d'exploitation, qui doit retrouver l'application à partir de l'extension du fichier. Cet exemple affiche un fichier PDF (que vous trouverez dans le zip téléchargeable).

```
rem Code19-01.sxw   bibli : Lancer Module1
Option Explicit

Sub AfficherPDF()
Dim sv As Object, fichier As String

sv = createUnoService("com.sun.star.system.SystemShellExecute")
fichier = "C:\Docs OpenOffice\ooo-1.1-cd-label.pdf"
sv.execute(fichier, "", 0)
End Sub
```

Nous invoquons le service `SystemShellExecute`, puis nous utilisons sa méthode `execute`. L'adresse du fichier doit être dans le format natif du système d'exploitation. La méthode `execute` peut aussi lancer une application *exécutable* en lui fournissant un argument, sous forme d'une chaîne de caractères :

```
sv.execute(application, argument, 0)
```

Toutefois, ceci ne fonctionne pas sur tous les systèmes (pour raison de sécurité). Le troisième argument de la méthode `execute`, s'il n'est pas nul, permet de ne pas afficher de message système si la méthode échoue. Il faut alors lui donner comme valeur la constante nommée :

```
com.sun.star.system.SystemShellExecuteFlags.NO_SYSTEM_ERROR_MESSAGE
```

La palette des couleurs

La palette des couleurs est accessible par l'interface utilisateur, menu Outils > Options > OpenOffice.org > Couleurs. Le contenu de la palette est utilisable grâce au service `ColorTable` de l'API. Cette macro affiche le nom de chacune des couleurs de la palette.

```
rem Code19-04.sxw   bibli : Standard Module1
option explicit

Sub PaletteCouleurs()
dim sv As Object, x As Long, lesCouleurs As Object

sv = createUnoService("com.sun.star.drawing.ColorTable")
' obtenir un tableau des couleurs de la palette
lesCouleurs = sv.ElementNames
for x = 0 to UBound(lesCouleurs()) ' lister les couleurs
  print x, lesCouleurs(x)
next
End Sub
```

La fonction Basic `createUnoService` nous renvoie un objet permettant d'utiliser le service dont le nom est donné en argument. L'objet `sv` possède une propriété `ElementNames`, qui est un tableau dont chaque élément est un nom de couleur.

Ce tableau est indexé à partir de zéro et la fonction `UBound` nous donne la valeur maximale d'index. Ceci nous permet de balayer le tableau avec une boucle `for` et d'afficher chaque nom de couleur.

ATTENTION

Les noms des couleurs prédéfinies sont localisés. Vous n'obtiendrez pas les mêmes noms sur une version allemande d'OpenOffice.org, par exemple.

La macro suivante montre comment utiliser une des couleurs de la palette.

```
rem Code19-04.sxw   bibli : Standard Module2
option explicit

Sub utiliserPalette()
dim sv As Object, c As Long, x As Long
sv = createUnoService("com.sun.star.drawing.ColorTable")

Const nomCouleur = "Sun 4" ' ceci est un exemple
if sv.hasByName(nomCouleur) then
  c = sv.getByName(nomCouleur) ' récupérer la valeur de couleur
  MsgBox("R = " & Red(c) & chr(13) & "V = " & Green(c) & chr(13) & "B = " & Blue(c))
else
  MsgBox(nomCouleur & " n'existe pas !", 16)
end if
End Sub
```

Pour simplifier, nous utilisons une constante pour `nomCouleur`. Nous aurions pu demander le nom à l'utilisateur, avec la fonction Basic `InputBox`.

Le service `ColorTable` expose la fonction `hasByName`, qui renvoie la valeur `True` si le nom en argument existe bien dans la palette. Dans ce cas, la fonction `getByName` du service nous donne la valeur correspondante. Nous avons vérifié l'existence du nom au préalable car, s'il n'était pas reconnu par `getByName`, une erreur serait déclenchée.

À partir de la valeur de couleur, nous affichons ses composantes avec les fonctions `Red`, `Green`, `Blue`. Les instructions `chr(13)` servent à changer de ligne dans le message affiché.

Le service `ColorTable` n'est pas documenté. Il comporte aussi des méthodes pour modifier la palette, mais qui ne fonctionnent pas.

Penser à l'utilisateur

Geler l'interface utilisateur

Certaines macros peuvent prendre un temps non négligeable pour s'exécuter et, si elles modifient le contenu du document, l'affichage va évoluer en même temps. Ce n'est pas agréable pour l'utilisateur et cela peut ralentir considérablement l'exécution. Il est possible de « geler » l'affichage sur l'interface utilisateur au début du traitement, puis de le libérer en fin de traitement, ce qui permet sa mise à jour automatique. Ceci est réalisé par deux méthodes de l'objet document.

```
monDocument.lockControllers
 ' ici l'affichage utilisateur est bloqué
monDocument.unlockControllers
 ' ici l'affichage utilisateur est libéré
```

En conséquence, si dans votre traitement vous souhaitez afficher une information à l'utilisateur, pensez à débloquer l'interface avant !

```
monDocument.unlockControllers
if MsgBox("Erreur rencontrée, continuer ?", 20) = 7 then
  ' réponse Non - terminer le programme
  ' - - - - -
else
  ' réponse Oui - continuer
  monDocument.lockControllers
end if
```

Pour chaque `lockControllers` exécuté, il faut exécuter plus tard un `unlockControllers`. À l'inverse, vous ne pouvez pas exécuter `unlockControllers` sans avoir auparavant exécuté `lockControllers`. En effet, l'API maintient un compteur de blocage, et non pas un simple indicateur binaire. Vous devez donc exécuter le même nombre de ces instructions, quels que soient les chemins parcourus à l'exécution du programme.

Empêcher les actions de l'utilisateur

Si l'affichage est gelé, l'utilisateur n'est pas pour autant empêché de pianoter sur le clavier et donc de modifier le document, bien que le résultat ne soit pas visible. Bien sûr, il est souhaitable de prévenir l'utilisateur quand un travail va prendre un certain temps, avec un message préalable et un indicateur de progression.

La manière la plus simple d'interdire à l'utilisateur d'effectuer des actions intempestives consiste à ouvrir un panneau de dialogue. Tant que le panneau reste ouvert, toute action sur le document est bloquée. L'utilisateur déclenchera le travail en cliquant sur un bouton de ce panneau ; la routine de gestion de l'événement pourra alors mettre à `False` la propriété `Enable` de la plupart des contrôles du dialogue, afin de lui interdire de les actionner. Il lui restera cependant encore la possibilité de fermer le dialogue en cliquant sur le X de fermeture de la fenêtre. Dans ce cas, le travail en cours continuera, mais l'utilisateur retrouvera son pouvoir de nuisance ; la fonction `Execute` du dialogue se terminera ensuite dès la fin d'exécution du sous-programme d'événement qui a déclenché le travail.

Il est parfaitement possible avec un dialogue de déclencher un traitement avec un bouton et de permettre à l'utilisateur de l'interrompre proprement en actionnant un deuxième bouton. La routine d'événement propre à ce deuxième bouton modifiera une variable commune, testée périodiquement pendant le traitement.

État d'avancement du travail

Nous avons vu dans les dialogues qu'il existe un contrôle Barre de progression. Il existe un autre moyen de rassurer l'utilisateur pendant un travail de durée non négligeable : le service `StatusIndicator` permet d'afficher un texte et une barre de progression en bas d'une fenêtre (`frame`) de document OpenOffice.org.

RESSOURCE

Le principe est décrit par Danny Brewer dans le forum *Code Snippets* (http://www.oooforum.org/forum/viewtopic.php?t=11200) et dans le chapitre 5.1 du document « Useful Macro Information For OpenOffice » d'Andrew Pitonyak, disponible sur Internet. Nous avons fait une synthèse de ces deux sources.

Notre exemple simule l'avancement du travail avec une boucle temporisée. En pratique, cette boucle serait remplacée par le codage permettant de réaliser le travail et on afficherait des messages plus utiles.

```
rem Code19-03.sxw   bibli : Standard Module1
Option Explicit

Sub IndiquerProgression()
Dim monDocument As Object, avance As Object, i As Long
monDocument = ThisComponent
monDocument.lockControllers
avance =  monDocument.CurrentController.StatusIndicator
avance.start("Attendez svp ...", 100)
For i = 1 To 100
  avance.Value = i
  if i = 75 then  avance.Text = "Patience, bientôt fini ..."
  Wait 80
Next
avance.Text = "Terminé !"
wait 800
avance.end
monDocument.unlockControllers
End Sub
```

Nous commençons par geler l'affichage sur l'interface utilisateur. Nous verrons en effet que cela n'empêche pas l'affichage de l'indicateur de progression. Puis, nous récupérons dans la variable `avance` l'objet `StatusIndicator` qui appartient au contrôleur courant, c'est-à-dire celui de la fenêtre du document où s'exécute la macro. La méthode `start` de l'objet `avance` sert à afficher la barre de progression accompagnée de son texte. Le deuxième argument est la valeur maximale de la progression. La barre de progression apparaît au bas de la fenêtre du document.

La valeur de progression est mise à jour avec la propriété `Value` de l'objet `avance`. Le texte associé peut être modifié par la propriété `Text` de cet objet. On pourra ainsi indiquer quelle phase du travail est en cours de réalisation.

En fin de travail, la barre de progression est supprimée en appelant la méthode `end` de l'objet `avance`. Aussitôt, le contenu habituel du bas de la fenêtre réapparaît. L'affichage est ensuite débloqué pour l'utilisateur.

Visibilité du document

Lorsque vous manipulez plusieurs documents, un seul sera au premier plan. Un document est affiché dans une fenêtre, que nous obtenons ainsi :

```
Dim doc1 As Object, fenetre1 As Object
fenetre1 = doc1.CurrentController.Frame.ContainerWindow
```

Il en est de même pour chaque document ; la méthode `toFront` de la fenêtre la fait passer à l'avant-plan par rapport aux autres fenêtres OpenOffice.org.

```
fenetre2.toFront
```

La méthode `toBack` renvoie la fenêtre au plus bas de la pile des fenêtres :

```
fenetre1.toBack
```

Si vous chargez un document de manière invisible, vous ne pourrez pas le rendre visible en cours de travail. Au contraire, un document chargé de manière visible peut être rendu invisible ou visible à volonté en changeant la propriété `Visible` de sa fenêtre :

```
fenetre2.Visible = False
'  modification invisible du document ...
fenetre2.Visible = True
```

Traitements spécifiques à MS-Windows

Accéder à la base de registres de MS-Windows

Didier Dorange-Pattoret est l'auteur initial de cette macro, qui lit des clés de la base de registres utilisée par MS-Windows pour afficher la version du système d'exploitation de l'ordinateur. Elle utilise des fonctions situées dans la bibliothèque `ImportWizard` de `soffice`.

```
Sub VersionValue
Const HKEY_LOCAL_MACHINE = &H80000002
Dim sVersion as String
Globalscope.BasicLibraries.LoadLibrary("ImportWizard")

OpenRegKey(HKEY_LOCAL_MACHINE, _
     "Software\Microsoft\Windows\CurrentVersion\")
sVersion = QueryValue(HKEY_LOCAL_MACHINE, _
     "Software\Microsoft\Windows\CurrentVersion\","ProductName")

If sVersion = "" Then
  OpenRegKey(HKEY_LOCAL_MACHINE, _
      "SOFTWARE\Microsoft\Windows_NT\CurrentVersion\")
  sVersion = QueryValue(HKEY_LOCAL_MACHINE, _
  "SOFTWARE\Microsoft\Windows NT\CurrentVersion\","ProductName")
End If
Print sVersion
End Sub
```

On en déduit comment accéder à d'autres clés de la base de registres (dont la structure dépend de la version de MS-Windows).

Utiliser l'API de MS-Windows

Certains développements peuvent nécessiter l'appel de fonctions internes de ce système d'exploitation. Que ce soit, comme dans la section précédente, pour accéder à la base de registres, pour utiliser d'autres fonctionnalités ou encore pour migrer un applicatif existant, Basic permet de les définir pour ensuite les appeler dans les macros comme des fonctions internes.

```
Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" _
 (ByVal hKey As Long, _
  ByVal lpSubKey As String, _
  ByVal ulOptions As Long, _
  ByVal samDesired As Long, _
  phkResult As Long) As Long

Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
 (ByVal lpClassName As long, _
  ByVal lpWindowName As String) As Long

Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
 (ByVal hwnd As Long, _
  ByVal wMsg As Long, _
  ByVal wParam As Long, _
  ByVal lParam As long) As Long
```

L'instruction `Declare` est tout à fait analogue à celle utilisée lors des appels d'`API win32` depuis d'autres langages comme Visual Basic™ et les habitués ne devraient pas être perturbés. Vous trouverez un exemple d'utilisation de ces fonctions dans la macro `InstallVirgule` disponible sur le site fr.openoffice.org. Cependant, l'opérateur `AddressOf` n'existe pas et vous ne pourrez donc pas utiliser d'instructions `Callback` nécessaires par exemple à l'implémentation de `Hooks`.

Cette technique permet également d'appeler toute fonction dans une DLL que vous auriez créée dans un autre langage.

> **REMARQUE** **Développement multi-plates-formes**
>
> Notez que l'utilisation de ces fonctionalités propres à un système d'exploitation brise la notion de multi-plates-formes de développements de macros OpenOffice.org.

Manipuler les objets COM

Étant dans le contexte particulier de l'utilisation d'OpenOffice.org uniquement sous environnement MS-Windows, il peut quelquefois être intéressant d'utiliser les technologies propres à ce système d'exploitation.

Toujours depuis nos macros, nous avons la possibilité d'utiliser un pont vers la technologie COM et utiliser les serveurs COM et activeX.

```
rem Code19-06.sxw   bibli : Standard Module1
'Article complet disponible sur
'http://www.oooforum.org/forum/viewtopic.php?t=9815

'Utilisation de MS Word depuis une macro OpenOffice.org
Sub loading_MSWord( )
Dim oleService As Object
Dim oword As Object
Dim odoc As Object

oleService = createUnoService("com.sun.star.bridge.OleObjectFactory")
oword = oleService.createInstance("Word.Application")
' l'objet va utiliser l'API de Word
oword.Visible = True
odoc = oword.Documents.Add
odoc.Range.Text = "Hello World!"
End Sub

'Utilisation d'Internet Explorer depuis une macro OpenOffice.org
Sub using_IE( )
Dim oleService
Dim IE
```

```
oleService = createUnoService("com.sun.star.bridge.OleObjectFactory")
IE = oleService.createInstance("InternetExplorer.Application.1")

' l'objet va utiliser l'API d'Internet Explorer
IE.Visible = 1
IE.Navigate("http://fr.openoffice.org")
End Sub
```

Le premier exemple va instancier MS-Word pour y écrire le fameux « Hello world !! ». Le deuxième exemple va lancer Internet Explorer pour l'URL d'OpenOffice.org.

Une fois le pont obtenu avec `com.sun.star.bridge.OleObjectFactory` et un objet spécifique créé avec le `queryInterface`, nous pouvons l'utiliser dans notre macro avec sa propre API comme le montre `oword.Documents.Add` ou bien `odoc.Range.Text`.

ATTENTION API différentes

Utiliser des API différentes dans un même programme nécessite une certaine rigueur pour ne pas mélanger les concepts. Les macros OpenOffice.org ont leurs propres API et philosophie, les objets instanciés les leurs. Le tout est de ne pas se mélanger les pinceaux et surtout de bien documenter votre code.

Si ces deux exemples ont un intérêt limité (sauf éventuellement à des fins de migration pour l'accès à Word), la méthodologie peut être intéressante sur des applicatifs spécifiques développés avec ces technologies.

À NOTER Pour aller plus loin

Le lecteur pourra se référer au chapitre 3.4.4 du « Developer's Guide », ainsi qu'au forum d'OpenOffice.org duquel sont issus ces deux exemples :

- http://www.oooforum.org/forum/viewtopic.php?t=9815

Envoyer un document par courrier électronique

Il existe deux services de l'API permettant d'envoyer un courrier électronique.

- `SimpleCommandMail` utilise l'application de messagerie fonctionnant en ligne de commande, si une telle application existe. Ceci ne semble pas fonctionner sous Windows.
- `SimpleSystemMail` utilise l'application de messagerie par défaut, ce qui aboutit à ouvrir une fenêtre de votre système de messagerie favori, montrant le message électronique pré-rempli, prêt à être envoyé.

Tous les champs courants peuvent être remplis, à l'exception du texte du message. Pour transmettre une information, on utilisera à la place le contenu d'un ou plusieurs fichier(s) attachés au courrier.

```
rem Code19-02.sxw   bibli : Mel Module1
Option Explicit

Sub EnvoyerE_Mail()
Dim sv As Object, unClient As Object, monMel As Object
Dim enCopie(1) As String, annexes(1) As String ' (exemples)

'sv = createUnoService("com.sun.star.system.SimpleCommandMail")
sv = createUnoService("com.sun.star.system.SimpleSystemMail")
unClient = sv.querySimpleMailClient
if IsNull(unClient) then
  MsgBox("Client de messagerie non disponible", 16)
  stop
end if
monMel = unClient.createSimpleMailMessage
with monMel ' évite de répéter monMel.xxxx
  .Recipient = "BillGates@microsoft.com"
  enCopie(0) = "alain@ici"
  enCopie(1) = "julie@ailleurs"
  .CcRecipient = enCopie()
  .Originator = "inconnu@neant.fr"
  .Subject = "Ceci est un test"
  annexes(0) = convertToURL("C:\Dezip\Icones\etoile_26.bmp")
  annexes(1) = convertToURL("C:\Docs OpenOffice\baratin.txt")
  .Attachement = annexes()
end with
unClient.sendSimpleMailMessage(monMel, 0)
End Sub
```

Le service est d'abord invoqué, puis on lui demande de chercher le client de messagerie (ce terme désigne l'application qui va envoyer le courrier électronique). La méthode `createSimpleMailMessage` du client nous renvoie un objet courrier vide. C'est une structure dont les éléments sont les champs principaux du courrier. Votre client de messagerie acceptera ou non que le champ `Originator` soit quelconque.

L'objet courrier est envoyé au client avec la méthode `sendSimpleMailMessage`. Le deuxième argument peut contenir un des indicateurs suivants, ou les deux, en les additionnant :

```
com.sun.star.system.SimpleMailClientFlags.NO_USER_INTERFACE
com.sun.star.system.SimpleMailClientFlags.NO_LOGON_DIALOG
```

Le premier indicateur envoie le courrier sans afficher la fenêtre du client de messagerie. Le deuxième indicateur n'affiche pas la fenêtre de connexion si l'utilisateur n'est pas connecté ; dans ce cas, une erreur se produit.

Utiliser un serveur Web

Comme nous l'avons vu précédemment, OpenOffice.org utilise des URL pour ouvrir les documents. Nous allons détourner ici cette fonctionnalité pour illustrer une utilisation possible d'une macro pour interagir avec un serveur Web.

Imaginons un serveur Web exposant un script PHP (ou autre) à l'URL http://monsite.com/lescript.php. Ce script attend un argument `prenom` et se charge d'afficher un message de bienvenue dans le navigateur. L'URL à utiliser sera donc : http://monsite.com/lescript.php?prenom=Toto

La macro suivante appelle le serveur, lui passe un prénom et y lit la réponse.

```
sub AppelURL()
  dim lePrenom as string
  dim monURL as string
  dim laReponse as string
  'On demande le Prénom
  lePrenom=InputBox("Quel est votre Prénom ?")

  'On construit l'URL
  monURL=ConvertToURL _
    ("http://monsite.com/lescript.php?prenom=" & lePrenom)

  'Ouverture de la communication
  Open monURL for input as #1
    'On récupère la réponse du serveur
    line input #1, laReponse
  close #1

  'On affiche le résultat
  MsgBox laReponse,0,"Réponse du serveur"

end sub
```

Comme nous le voyons, l'URL est reconstruite par simple concaténation de chaînes de caractères. La fonction `ConvertToURL` est utilisée pour prendre en charge d'éventuels caractères spéciaux dans la variable `lePrenom`. Comme la macro doit lire la réponse affichée par le serveur, nous utilisons l'instruction `open ... for input` sur l'URL.

Le fait d'utiliser `monURL` comme nom de fichier a le même effet que de l'appeler depuis un navigateur : le script va être interprété et la réponse du serveur produite. Ainsi, dès l'ouverture du fichier, le serveur Web est sollicité.

Pour lire la réponse, il suffit d'utiliser l'instruction `line input` comme on le ferait sur un véritable fichier. L'exemple utilisé sous-entend qu'il n'y a qu'une seule ligne à lire. Une boucle peut être utilisée si ce n'est pas le cas.

Cet exemple simpliste permet d'ouvrir l'horizon à toutes sortes d'interactions si, par exemple, le serveur est prévu pour agréger des données ou s'il est couplé à une base de données.

Intercepter un événement (listener)

Nous savons qu'une macro peut être déclenchée à l'ouverture d'un document, ou à sa fermeture, etc. Nous avons vu dans les boîtes de dialogue et dans les formulaires comment certains événements peuvent être interceptés par une macro OOoBasic. Cette affectation d'une macro OOoBasic à un événement via l'interface utilisateur n'est qu'un sous-ensemble de tous les événements susceptibles d'être analysés. Le mécanisme d'interception d'événements (en anglais *listener*, signifiant écouteur) offert par l'API est un moyen plus général, donnant accès à un grand nombre d'événements.

La seule source d'information disponible sur les différents événements est l'API. Pour intercepter des événements non accessibles par les moyens ordinaires, il vous faudra étudier les pages de l'API concernant un objet donné et chercher les interfaces dont le nom se termine par `Listener`.

Par exemple, supposons qu'un objet `Machin` prenne en charge une interface imaginaire `XBlablaListener` située dans la branche `com.sun.star.truc` de l'API. Lorsque vous lirez la page correspondante de la documentation, vous trouverez par exemple trois noms de fonction, correspondant chacune à un événement particulier :

- `debuterTravail` ;
- `faireQuelqueChose` ;
- `faireAutreChose`.

Pour intercepter les événements de `Blabla`, vous devez vous inscrire comme écouteur (comme à la fac). On utilise pour cela, d'une part la fonction Basic `CreateUnoListener` qui renvoie un objet spécifique, d'autre part une méthode de l'objet `Machin`, destinée à réaliser la « mise sur écoute ».

```
Dim ecouteur As Object
Dim Machin As Object
' (on suppose que Machin est déjà obtenu)

ecouteur = CreateUnoListener("monGrainDeSel_",
                    "com.sun.star.truc.XBlablaListener")
Machin.addBlablaListener(ecouteur)
```

Le mécanisme d'interception est maintenant en place. Pour chaque événement concernant l'interface `Blabla` de l'objet `Machin`, il va exécuter une macro. Le *nom de la macro* se compose :

- du premier argument de `CreateUnoListener`, ici ce sera `monGrainDeSel`,
- suivi du nom de l'événement, par exemple `blablaDebute`.

PIÈGE Macros d'interception

Le mécanisme d'interception essaye aveuglément d'exécuter une macro dont le nom dépend de l'événement. Si elle n'existe pas, vous aurez une erreur Basic qui ne vous précisera pas quel est le nom manquant. Cela signifie que vous devez avoir écrit une macro pour chacun des événements pris en charge, même si vous ne vous intéressez pas à certains des événements.

Dans notre exemple, vous devrez avoir *quatre* macros :

```
Sub monGrainDeSel_debuterTravail()
End Sub

Sub monGrainDeSel_faireQuelqueChose()
End Sub

Sub monGrainDeSel_faireAutreChose()
End Sub

Sub monGrainDeSel_disposing()
End Sub
```

La dernière macro concerne un événement toujours présent dans les `Listener`. Il peut être appelé en dehors de toute exécution de macro, en particulier quand le document va être fermé. D'une manière générale, vous devez traiter les événements hérités par un `Listener`.

La structure exacte de chaque macro dépend de l'interface et de l'événement. En général, l'événement transmet une information que vous pouvez obtenir en premier argument de la macro. Certains événements offrent à l'écouteur la possibilité de les approuver ou non (droit de veto). Pour ceux-là, le sous-programme est une fonction renvoyant une valeur `Boolean`. Une valeur `True` accepte l'événement, une valeur `False` le supprime.

Un gestionnaire d'événement est supprimé avec la méthode `removeXxxListener` de l'objet écouté :

```
Machin.removeBlablaListener(ecouteur)
```

Il faut placer la mise sur écoute et la fin de la mise sur écoute au bon moment, compte tenu que vous ne pouvez pas tout maîtriser, par exemple un formulaire qui se connecte à

une base de données et déclenche ainsi des événements dès l'ouverture du document, ainsi qu'à la fermeture du document.

Plusieurs programmes peuvent greffer un gestionnaire d'événement sur le même ensemble d'événements. Par exemple, les événements interceptables avec l'onglet Événements des contrôles de boîte de dialogue ou de formulaire sont eux-mêmes traités par un gestionnaire d'événement interne. Lorsque vous approuvez un événement, il se peut qu'un autre gestionnaire soit appelé ensuite, qui donnera son veto. Un événement a lieu seulement si aucun gestionnaire ne s'y oppose.

Pour un premier abord des `Listener`, vous pouvez regarder le jeu `leBOOolier.sxd` disponible sur le site fr.openoffice.org. Un `Listener` sur la frappe de touche dans un document Draw est mis en place pour contrôler l'appui des flèches de direction afin de déplacer un objet Draw. Nous reproduisons ici ce traitement spécifique.

```
Sub RegisterKeyHandler
  oDocView = ThisComponent.getCurrentController
  oKeyHandler = createUnoListener _
                ("MonJeu_", "com.sun.star.awt.XKeyHandler")
  oDocView.addKeyHandler(oKeyHandler)
End Sub

'----------------------------------------
Sub UnregisterKeyHandler
  on error resume next
  oDocView.removeKeyHandler(oKeyHandler)
End Sub
'----------------------------------------

Function MonJeu_KeyPressed(oEvt) as Boolean
  select case oEvt.keyCode
   case 1026: 'Fleche Gauche
      if not isPause then
         bougePanier=-1
         call Deplacepanier(bougePanier)
      endif
   case 1027: 'Fleche droite
      if not isPause then
         bougePanier=1
         call Deplacepanier(bougePanier)
      endif
   case 1281: 'Esc
      arret=true
   case 527: 'Pause
      call TooglePause
   case else:
      ' autres touches
  end select
```

```
  MonJeu_KeyPressed=true
End Function

'----------------------------------------
Function MonJeu_KeyReleased(oEvt) As Boolean
  MonJeu_KeyReleased = False
End Function
```

L'utilisation d'un `Listener` se fait en deux étapes. Il faut d'abord le préparer. C'est ce que fait la routine `RegisterKeyHandler` en utilisant la fonction `CreateUnoListener`.

Comme nous l'avons vu, cette fonction a besoin de deux arguments, le premier étant le préfixe des fonctions d'événement qui seront appelées, le deuxième le type du `Listener` que nous trouverons dans le module `com.sun.star.awt` de l'API.

On y trouvera des `Listener` concernant le clavier, la souris, les menus, dont certains sont énumérés dans le tableau 19-2.

Tableau 19–2 Quelques Listener

com.sun.star.awt.	Signification
`XKeyHandler`	Gère un événement de type clavier KeyEvent - Peut bloquer l'événement
`XKeyListener`	Gère un événement de type clavier KeyEvent
`XMenuListener`	Gère un événement de type menu MenuEvent
`XMouseClickHandler`	Gère un événement de type souris MouseEvent - Peut bloquer l'événement
`XMouseListener`	Gère un événement de type souris MouseEvent

Chaque `Listener` est associé à un type d'événement qui est en fait une structure également disponible dans le module `com.sun.star.awt`. Nous ne nous attacherons qu'à l'événement `KeyEvent`.

Le `Listener` est alors déclaré par la méthode `addKeyHandler` sur le document. La routine `UnregisterKeyHandler` permet de retirer ce `Listener` avec la méthode `removeKeyHandler`.

Une fois un `Listener` déclaré, tout événement le concernant appellera la méthode concernée associée au préfixe.

Ainsi, pour notre `Listener` de clavier, nous devons définir les fonctions `MonJeu_KeyPressed` et `MonJeu_KeyReleased`.

Chacune de ces fonctions accepte en argument d'entrée un objet de type `KeyEvent` renseignant sur le contexte dans lequel l'événement est intervenu (tableau 19-3).

Tableau 19–3 L'événement KeyEvent

Information disponible	Signification
`KeyCode`	Code numérique unique - `com.sun.star.awt.Key`
`KeyChar`	Caractère unicode transmis ou 0
`KeyFunc`	Constante de fonction - `com.sun.star.awt.KeyFunction`
`Modifiers`	État des touches mortes (Shift, Control et Alt) - `com.sun.star.awt.KeyModifier`
`Source`	Objet ayant engendré l'événement

Notre jeu ne s'intéresse ici qu'au code de la touche afin de pouvoir réagir sur certaines. Ces codes numériques peuvent facilement être obtenus en ajoutant une ligne dans la macro lors de la phase de test.

```
print oEvt.KeyCode
```

Néanmoins, les touches sont pour la plupart disponibles dans le groupe de constantes `com.sun.star.awt.Key`.

Enfin, nous constatons que ces fonctions sont de type booléen. Ceci est important puisqu'il s'agit en fait de traiter un `Handler` qui va pouvoir bloquer l'événement en évitant qu'il soit transmis au document. Si la fonction retourne `True`, l'événement est bloqué et n'est pas transmis. Si la fonction retourne `False`, l'événement est transmis au document.

Rappelons que même si aucun traitement n'est envisagé dans une fonction d'événement d'un `Listener`, il faut quand même la déclarer dans le programme.

Modifier des macros par programmation

L'écriture dynamique de macros

La programmation courante (à 99 %) consiste à concevoir des séquences d'instructions, les écrire au clavier, les sauvegarder dans un fichier, puis exécuter ce fichier. Dans certaines situations, il est intéressant de créer une série d'instructions « à la volée », par programme, et de les incorporer dans l'algorithme de ce même programme. Ceci est parfaitement possible en OOoBasic, avec une petite pirouette.

> Nous avons déjà présenté dans les chapitres 3 et 7 le concept des bibliothèques et des conteneurs de bibliothèques.

Un conteneur de bibliothèque, par exemple `BasicLibraries`, expose plusieurs méthodes permettant de modifier son contenu :

- `hasByName("bibliX")` renvoie `True` si la bibliothèque `bibliX` existe dans le conteneur.
- `getByName("bibliX")` renvoie l'objet bibliothèque correspondant.
- `removeLibrary("bibliX")` supprime la bibliothèque en argument.
- `createLibrary("bibliX")` crée la bibliothèque en argument et renvoie l'objet correspondant ; attention, cette bibliothèque ne contient encore aucun module.

Les méthodes spécifiques des conteneurs de bibliothèques sont documentées dans la branche `com.sun.star.script` aux interfaces :

- `XLibraryContainer`,
- `XLibraryContainer2`,
- `XLibraryContainerPassword`.

De manière très similaire à un conteneur de bibliothèques, une bibliothèque est elle-même un conteneur de modules. Une bibliothèque expose un tableau `ElementNames` listant les noms de modules qu'elle contient. Plusieurs méthodes permettent de gérer les modules d'une bibliothèque :

- `hasByName("moduleX")` renvoie `True` si le module `moduleX` existe dans la bibliothèque.
- `getByName("moduleX")` renvoie une chaîne de caractères qui est le texte complet du module.
- `removeByName("moduleX")` supprime le module en argument dans la bibliothèque.
- `insertByName("moduleX", texteModule)` ajoute le module `moduleX` dans la bibliothèque, le deuxième argument étant une chaîne de caractères contenant le texte complet du module.
- `replaceByName("moduleX", texteModule)` remplace le contenu du module `moduleX`.

Avec ces méthodes, il est assez aisé de créer une bibliothèque, puis de lui ajouter un module ou de modifier ce dernier. Dédiez une bibliothèque à votre module dynamique. Construisez dans ce module une routine `Sub` ou `Function` et appelez la routine depuis le reste du programme (qui reste fixe).

Un exemple plus complexe se trouve dans l'outil Xray, dont nous reparlerons à l'annexe C. La bibliothèque dynamique s'appelle `XrayDyn`. Son unique module contient plusieurs routines dont le contenu est dynamique, ce qui a conduit à systématiser la manière de le modifier. Voyez pour cela dans la bibliothèque `Xray`, module `_Utilities`, les macros `getInfo`, `InsertInfo`, `getModuleText`, `saveModuleText`. Des exemples d'utilisation se trouvent dans le module `Mod3` de la bibliothèque `Xray`.

Modifier les macros d'un autre document

Si vous devez effectuer des modifications systématiques dans des bibliothèques de macros d'une série de documents, la méthode manuelle consiste à ouvrir chaque document, puis chaque bibliothèque, puis chaque module de bibliothèque, pour effectuer la modification si nécessaire, sauver et fermer le document : tâche particulièrement ennuyeuse.

Nous allons, avec une macro, ouvrir un document et rechercher dans chaque module de ses bibliothèques de macros l'apparition d'un mot. S'il est présent, nous ajouterons un commentaire en tête du module. Il suffirait d'ajouter une boucle pour modifier ainsi toute une série de documents.

```
rem Code19-05.sxw   bibli : Standard Module1
Option Explicit

Sub ModifMacros()
Dim monDocument As Object, bLib As Object, lesBiblis As Object
Dim uneBibli As Object, lesModules As Object
Dim adresseDoc As String, nomBibli As String
Dim nomModule As String, codageModule As String
Dim motRecherche As String, liste As String
Dim x As Long, y As Long, modif As Boolean
Dim propFich(0) As New com.sun.star.beans.PropertyValue

motRecherche = "servGrad"
propFich(0).Name = "Hidden"
propFich(0).Value = True
adresseDoc = convertToURL("C:\Docs OpenOffice\Tata.sxd")
monDocument = StarDesktop.LoadComponentFromURL(_
              adresseDoc, "_blank", 0, propFich())
if IsNull(monDocument) then
  MsgBox("Le document n'existe pas", 16)
  stop
end if
modif = false
bLib = monDocument.BasicLibraries
lesBiblis = bLib.ElementNames
for x = 0 to UBound(lesBiblis)
  nomBibli = lesBiblis(x)
  liste = liste & "Bibliothèque " & nomBibli & chr(13)
  uneBibli = bLib.getByName(nomBibli)
  ' charger la bibliothèque pour voir le contenu de ses modules
  bLib.LoadLibrary(nomBibli)
  lesModules = uneBibli.ElementNames

  for y = 0 to UBound(lesModules)
    nomModule = lesModules(y)
    liste = liste &  "Module " & nomModule
    codageModule = uneBibli.getByName(nomModule)
```

```
    if InStr(codageModule, motRecherche) > 0 then
      ' modification du module : ajout d'un commentaire
      codageModule = "REM --- modifié ---" & _
                     chr(13) & chr(10) & codageModule
      uneBibli.replaceByName(nomModule, codageModule)
      modif = True
      liste = liste & "   mot trouvé" & chr(13)
    else
      liste = liste & "   mot PAS trouvé" & chr(13)
    end if
  next
next
MsgBox(liste)
if modif then monDocument.Store
on Error Resume Next
monDocument.close(True)
On Error GoTo 0
End Sub
```

Pour un effet plus spectaculaire, faites une copie du document `Code13-02.sxd` du zip téléchargeable, qui contient plusieurs bibliothèques. C'est le document désigné par `Tata.sxd` dans notre macro. Nous allons rechercher le mot `servGrad` dans les modules de bibliothèques.

Le point important est d'utiliser le conteneur `BasicLibraries` du document à modifier, au lieu du conteneur `BasicLibraries` ordinaire, qui est celui du document qui a lancé la macro. Le reste du codage utilise les notions indiquées à la section précédente. La chaîne de caractères `liste` mémorise les bibliothèques et les modules trouvés, avec le résultat de recherche.

Particularités apportées par OOo 2.0

Résultat du Dispatcher

Comme indiqué à plusieurs endroits de ce livre (chapitre 7, annexe A, annexe B), il est parfois utile d'utiliser les fonctions fournies par le *Dispatcher*.

À partir de la version 2 d'OpenOffice.org, il est possible de savoir si la fonction a pu être lancée par le Dispatcher : ce dernier renvoie une structure `com.sun.star.frame.DispatchResultEvent` décrite au tableau 19-4.

Tableau 19–4 Structure de DispatchResultEvent

Propriété	Type	Signification
`State`	`Integer`	Efficacité du Dispatcher, constante nommée
`Result`	`Variant`	Résultat éventuel de la fonction appelée

L'élément `State` indique si le Dispatcher a pu lancer la fonction. Les constantes nommées sont de la forme :

```
com.sun.star.frame.DispatchResultState.SUCCESS
```

Elles sont décrites dans le tableau 19-5.

Tableau 19–5 Constantes State du Dispatcher

Constante	Signification
FAILURE	Fonction lancée
SUCCESS	La fonction n'a pu être lancée
DONTKNOW	Impossible de savoir si la fonction a été lancée

Le contenu de l'élément `Result` dépend entièrement de la fonction lancée. Si elle ne renvoie pas de résultat, `Result` vaut `Empty`.

Appeler un script écrit dans un autre language

Il est possible de lancer des scripts créés dans d'autres langages que Basic, et même de lancer un script Basic (normalement depuis un autre langage, car depuis Basic, ceci n'a pas d'intérêt). Dans cet exemple, on lance un script simple qui ne nécessite de transférer aucune information vers ou depuis le script appelé.

```
rem Code19-07.odt  bibli : Standard Module1
Option Explicit

Sub lancerScripts()
' il n'existe pas de HelloWorld Basic, appel d'un autre script
simpleScript("Gimmicks.AutoText.Main",  "Basic", "application")

Dim nouvDoc As Object
nouvDoc = StarDesktop.loadComponentFromURL( _
          "private:factory/swriter", "_blank", 0, Array() )
' ces scripts vont écrire sur le document Writer
simpleScript("HelloWorld.helloworld.js",   "JavaScript", "share")
simpleScript("HelloWorld.HelloWorld.printHW","Java",     "share")
simpleScript("HelloWorld.helloworld.bsh",  "BeanShell",  "share")
simpleScript("HelloWorld.py$HelloWorldPython","Python",  "share")
MsgBox("Cliquez OK pour fermer le document")
nouvDoc.close(True)
End Sub
```

```
Sub simpleScript(nomScript As String, langage As String, _
                                      emplacement As String)
Dim mspf As Object, scriptPro As Object, monScript As Object

mspf = createUnoService( _
        "com.sun.star.script.provider.MasterScriptProviderFactory")
scriptPro = mspf.createScriptProvider("")
On Error Goto PasScript1
monScript = scriptPro.getScript("vnd.sun.star.script:" & nomScript & _
               "?language=" & langage & "&location=" & emplacement)
' appel de script simple, sans arguments et sans résultat en retour
monScript.invoke(Array(), Array(), Array())
On Error Goto 0
Exit Sub

PasScript1:
Resume PasScript2
PasScript2:
MsgBox("Script pas trouvé : " & nomScript, 16)
End Sub
```

Nous avons lancé des scripts qui sont fournis d'origine, dans la section « Macros OpenOffice.org », c'est-à-dire dans la branche `share` de l'installation.

Notre exemple est en OOoBasic, mais on aurait pu écrire l'équivalent dans un des autres langages de script reconnus.

Pour éviter l'aspect « formule magique » de l'URI d'appel du script, nous l'avons décomposée en ses éléments fondamentaux. La syntaxe de l'argument `nomScript` est propre à chaque langage, vous la devinerez en comparant l'exemple avec l'emplacement réel du script appelé. Un autre moyen est de mettre sur un document vierge un bouton de formulaire, et d'affecter à un de ses événements le script souhaité. L'URI complète est alors visible dans le panneau d'assignation. Attention, la casse des caractères doit être respectée.

Pour un script différent de Basic, l'argument `emplacement` peut prendre les valeurs :

- `user` si le script est dans « Mes Macros »,
- `share` si le script est dans « OpenOffice.org Macros »,
- `document` si le script est dans le même document que le lanceur du script.

Pour un script Basic, l'argument `emplacement` peut prendre les valeurs :

- `application` si le script est dans « Mes Macros » ou dans « OpenOffice.org Macros »,
- `document` si le script est dans le même document que le lanceur du script.

Il n'est pas possible, à partir d'un document, de lancer un script appartenant à un autre document.

Vous trouverez un exemple avec transmission de paramètres sur le site OOoForum, section Code Snippets : Calling JavaScript from Basic: regular expressions http://www.ooo-forum.org/forum/viewtopic.phtml?t=21564

> **À NOTER Pour aller plus loin**
>
> Vous trouverez plus de détails (en anglais) dans le SDK 2.0, Developer's Guide, chapitre 18.4.
>
> La documentation IDL de la méthode invoke vous indiquera comment sont transmis les données entre le script et l'appelant. Elle se trouve à l'interface : `com.sun.star.script.provider.XScript`

Conclusion

Gestion de fichiers ZIP, connexion à Internet ou envoi de courrier électronique, la liste ne peut être exhaustive. Le chapitre qui s'achève illustre la puissance mise à disposition des macros par l'intermédiaire de l'API. Nul besoin d'ajouter un quelconque composant, les outils sont directement accessibles et à la disposition du concepteur de macros. Ce chapitre clôt l'exposé et l'illustration des macros et de l'API à proprement parler.

L'ensemble de l'ouvrage étudié, les auteurs espèrent que le lecteur aura acquis les méthodes nécessaires à la rédaction de « belles macros » mais aussi une certaine méthodologie de recherche des informations au sein de la documentation de l'API. Il pourra également s'appuyer sur les annexes de cet ouvrage.

Outils et ressources

Il nous a paru indispensable de compléter cet ouvrage par une annexe expliquant ce qu'est l'API et comment en obtenir des informations pour aller encore plus loin. Nous présentons ensuite une liste de routines utilitaires. Nous signalons également les ressources Internet incontournables pour qui souhaite se tenir à jour : il s'agit de forums où chercher assistance, ou de sites fournissant des exemples de macros, des documents explicatifs et des outils.

A

Comprendre l'API d'OpenOffice.org

Cette annexe offre une introduction à la principale source d'information officielle sur les mécanismes internes à OpenOffice.org. Tous ces documents, en anglais, sont accessibles à un utilisateur confirmé.

Qu'est-ce que l'API ?

L'API (Application Programming Interface) d'OpenOffice.org est un ensemble de points d'entrée permettant de manipuler OpenOffice.org – quoique ne couvrant cependant tous ses mécanismes. Sans être liée à un langage de programmation particulier, elle est accessible de manière privilégiée avec OOoBasic, mais on peut aussi utiliser des langages tels que Java, Python, Delphi, voire d'autres outils de script tel VBscript.

L'API est un système logiciel très complexe par son étendue et par ses concepts ; elle est composée de très nombreux objets héritant les uns des autres. Le développeur la verra comme un ensemble de méthodes, de fonctions, de propriétés, de structures de données. Elle est organisée en un arbre dont la racine est :

```
com.sun.star.
```

À partir de cette racine, elle se subdivise en modules, qui sont des groupements de logiciels, par exemple :

```
com.sun.star.text.
```

Tout élément de l'API se situe à un nœud de cet arbre, en juxtaposant les branches successives :

```
com.sun.star.text.WrapTextMode.PARALLEL
```

Un module se décompose en un service ou plusieurs, et parfois en sous-modules, comme pour le module `text`.

Un service comporte lui-même parfois des services (en anglais *included services*). Il peut avoir des propriétés. Ce sont des « vraies » propriétés (nous verrons plus loin qu'OOoBasic présente aussi des pseudo-propriétés). Un service possède en général une interface ou plusieurs. Tous les noms d'interfaces commencent par un X (majuscule). La documentation dit que le service *exporte* des interfaces, ce qui signifie qu'il les met à la disposition du programmeur.

Une interface contient des méthodes. Ce sont des sous-programmes utilisables à partir de l'objet considéré. Les méthodes utilisent des arguments et renvoient éventuellement un résultat (sous-programme fonction). Chaque argument et chaque résultat peut être :

- un objet API,
- une donnée d'un type simple (booléen, entier, flottant),
- une séquence de données ou d'objets (présenté en OOoBasic sous forme de tableau),
- une constante nommée, qui est un type entier dont les valeurs possibles sont définies avec des noms qualifiés,
- une structure de données, qui est un regroupement de données accessibles individuellement.

Comme l'API est indépendante du langage, le type simple indiqué pour une donnée doit être « traduit » dans le type de données le plus proche pour le langage de programmation utilisé. Cela peut poser quelques difficultés : par exemple, OOoBasic ne possède ni type `Byte`, ni type `hyper`.

Un objet API, en dehors des constantes et des structures, comporte un service ou plusieurs. Il y a lieu de distinguer les services pris en charge, directement utilisables, et les services disponibles, qui peuvent être invoqués avec la fonction `createInstance`.

L'API réelle et l'API selon OOoBasic

OOoBasic a plusieurs avantages par rapport aux autres langages :

- Il est le plus intégré à l'application.
- Il est facile à apprendre.
- Il est conçu pour simplifier l'accès aux primitives de l'API : il n'impose pas de respecter les majuscules et minuscules dans l'emploi des noms de routines et de propriétés.
- Il connaît les valeurs des constantes nommées.
- Il permet d'utiliser directement les interfaces d'un objet, contrairement à Java™.
- Il permet d'utiliser comme une pseudo-propriété le couple de méthodes `get` et `set` manipulant la même donnée interne, contrairement à Java.

Pour toutes ces raisons, les exemples en Java, nombreux dans la documentation, sont bien plus difficiles à lire que leurs équivalents en OOoBasic. À titre d'exemple, voici un extrait de code en Java qui modifie un mot dans un texte Writer et modifie le curseur pour écrire en gras.

```
mxDocCursor = mxDocText.createTextCursor();
XSentenceCursor xSentenceCursor = (XSentenceCursor)
 UnoRuntime.queryInterface(XSentenceCursor.class, mxDocCursor);
xSentenceCursor.gotoNextSentence(false);
XWordCursor xWordCursor = (XWordCursor)
 UnoRuntime.queryInterface(XWordCursor.class, mxDocCursor);
xWordCursor.gotoNextWord(false);
xWordCursor.gotoNextWord(true);
mxDocText.insertString(xWordCursor, "hello ", true);
XPropertySet xCursorProps = (XPropertySet)
 UnoRuntime.queryInterface(XPropertySet.class, mxDocCursor);
xCursorProps.setPropertyValue("CharWeight",
 new Float(com.sun.star.awt.FontWeight.BOLD))
```

Ce code Java nécessite trois variables supplémentaires pour gérer le curseur, une pour chaque interface nécessaire. Voici pour comparaison le code OOoBasic équivalent, qui n'utilise que la variable curseur :

```
mxDocCursor = mxDocText.createTextCursor
mxDocCursor.gotoNextSentence(false)
mxDocCursor.gotoNextWord(false)
mxDocCursor.gotoNextWord(true)
mxDocText.insertString(mxDocCursor, "hello ", true)
mxDocCursor.CharWeight = com.sun.star.awt.FontWeight.BOLD
```

Lorsqu'il existe deux méthodes complémentaires simples, l'une servant à affecter une valeur à une donnée interne, l'autre servant à obtenir la valeur de cette donnée, OOoBasic

combine les deux sous la forme d'une pseudo-propriété, utilisable comme une simple variable. Voici deux codages OOoBasic qui réalisent exactement la même chose :

```
Dim Taille As Object
' codage autorisé par OOoBasic
Taille = dessin.Size
Taille.Width = Taille.Width * 2
dessin.Size = Taille

' application stricte de l'API
Taille = dessin.getSize()
Taille.Width = Taille.Width * 2
dessin.setSize(Taille)
```

L'utilisateur OOoBasic ne voit qu'une propriété `Size`. En réalité, cette propriété n'existe pas, mais elle est un raccourci vers deux méthodes :

- `setSize(valeur)` qui modifie la taille (en anglais *size*) de l'objet ;
- `getSize()` qui renvoie la taille de l'objet.

Pour retrouver dans l'API la description d'une propriété d'objet, il est donc nécessaire d'ajouter `get` ou `set` s'il s'agit d'une pseudo-propriété. Certaines données internes `Xyz` peuvent être manipulées seulement par `getXyz`, ou seulement par `setXyz`. Dans ces cas, la pseudo-propriété est restreinte à la seule lecture ou écriture.

On accède aux vraies propriétés des objets de l'API de manière plus simple en OOoBasic que dans d'autres langages comme Java. Exemple :

```
' codage autorisé par OOoBasic
couleur = UneCellule.CellBackColor
UneCellule.CellBackColor = RGB(255,255,204)

' deuxième manière, plus complexe, et aussi valide en OOoBasic
' ici la casse de CellBackColor doit être respectée
couleur =  UneCellule.getPropertyValue("CellBackColor")
UneCellule.setPropertyValue("CellBackColor", RGB(255,255,204))
```

Un objet dans une collection est accessible en OOoBasic par une indexation, comme si la collection était un tableau. En réalité, OOoBasic fait appel à la fonction `getByIndex` de la collection :

```
' codage autorisé par OOoBasic
uneFeuille = monDocument.Sheets(1)

' deuxième manière, plus complexe, et aussi valide en OOoBasic
uneFeuille = monDocument.Sheets.getByIndex(1)
```

Il existe aussi un raccourci OOoBasic pour accéder par son nom à un objet de collection. Nous ne l'avons pas employé car il peut donner des expressions ambiguës.

```
' codage recommandé
uneFeuille = monDocument.Sheets.getByName("Total")

' deuxième manière, valide en OOoBasic, déconseillée
uneFeuille = monDocument.Sheets.Total
```

ALLER PLUS LOIN **Liaison Basic - UNO**

L'interfaçage entre OOoBasic et UNO (le concept logiciel d'OpenOffice.org) est décrit plus longuement dans le « Developer's Guide », chapitre 3.4.3.

Les fonctions Basic dédiées à l'API

Tout au long de cet ouvrage, nous avons utilisé plusieurs fonctions OOoBasic qui facilitent l'accès à l'API. En voici une liste plus systématique.

StarDesktop

Il s'agit d'un objet prédéfini qui est le service de base d'OpenOffice.org. Il est un raccourci pour :

```
Dim monOOo As Object
monOOo = createUnoService("com.sun.star.frame.Desktop")
```

ThisComponent

Représente l'objet document en cours ; il est en général équivalent à :

```
Dim monDocument As Object
monDocument = StarDesktop.CurrentComponent
```

Cependant, si l'EDI est en premier plan, `CurrentComponent` renverra l'EDI et la macro ne fonctionnera pas, alors que `ThisComponent` continue à renvoyer l'objet document. C'est pourquoi `ThisComponent` est préférable.

Notez que `ThisComponent` n'est pas une variable, mais un appel de fonction : s'il y a plusieurs documents OpenOffice.org ouverts, elle renvoie le document OpenOffice.org dont la fenêtre est actuellement en avant-plan. C'est pourquoi il est préférable de ne l'appeler qu'au début de la macro et de sauver le résultat dans une variable interne.

CreateUnoStruct

Sert à obtenir une structure UNO, par exemple :

```
Dim uneProp As Object
uneProp = CreateUnoStruct("com.sun.star.beans.Property")
```

Il est plus simple de définir directement la variable :

```
Dim uneProp As New com.sun.star.beans.Property
```

CreateUnoValue

Sert à créer une donnée quelconque pour la transmettre à l'API. Cette méthode est prévue pour des problèmes très particuliers de conversion Basic-UNO.

CreateUnoService

Permet d'obtenir un objet capable de fournir le service donné en argument :

```
Dim demandePasse As Object
demandePasse = CreateUnoService( _
                   "com.sun.star.task.InteractionHandler")
```

Cette fonction est un raccourci pour :

```
Dim outilService As Object, demandePasse As Object
outilService = GetProcessServiceManager
demandePasse = outilService.createInstance( _
                   "com.sun.star.task.InteractionHandler")
```

CreateUnoListener

Permet à un programme de s'enregistrer comme auditeur d'un ensemble d'événements. Voir le principe au chapitre 19 et un exemple au chapitre 17.

CreateUnoDialog

En fait, cette fonction n'est concevable que dans un environnement OOoBasic, puisqu'elle sert à créer une boîte de dialogue (voir chapitre 15).

GetProcessServiceManager

Renvoie un objet permettant d'obtenir un service initialisé, soit par défaut, soit avec des arguments. Le premier cas est réalisé plus simplement avec `CreateUnoService`. Le deuxième cas est de la forme :

```
Dim outilService As Object, unService As Object
Dim args(1) ' un ou plusieurs arguments
outilService = GetProtcessServiceManager
' - initialiser le tableau args() avant cette instruction -
unService = outilService.createInstanceWithArguments( _
                  "com.sun.star.xxx.yyy.zzz", args())
```

Il existe aussi une troisième forme d'initialisation :

```
createInstanceWithArgumentsAndContext
```

Ces deux dernières formes ne sont utilisées que dans des cas assez particuliers.

IsUnoStruct

Renvoie `True` si la variable est une structure UNO. Exemple de structure UNO : le descripteur renvoyé par `createReplaceDescriptor` (voir chapitre 11). Cette fonction permet de distinguer une structure d'un objet véritable, ou d'une donnée simple.

Les variables de structure UNO sont de vraies valeurs, et non des références comme les variables sur les objets. C'est la raison des recopies nécessaires pour effectuer une modification, comme ici :

```
dim rognure as object
rognure = monImage.GraphicCrop
rognure.Bottom = -2000
monImage.GraphicCrop = rognure
```

EqualUnoObjects

Les variables représentant des objets sont en fait des références sur l'objet lui-même. Il est donc possible d'avoir deux variables pointant sur le même objet. Cette fonction permet de le vérifier.

HasUnoInterfaces

Renvoie `True` si l'objet en premier argument prend en charge toutes les interfaces des arguments suivants. Cette fonction est parfois utile pour rechercher un objet d'un type particulier dans une collection. Un autre moyen est d'utiliser la fonction `supportsService` des objets pour vérifier s'ils prennent en charge un service particulier.

Comprendre les messages d'erreur OOoBasic

Les messages d'erreur d'exécution de OpenOffice.org Basic sont souvent assez peu compréhensibles. Voici quelques informations pour vous aider.

Évitez les erreurs

Vérifiez la syntaxe de chaque module que vous avez écrit ou modifié. Il suffit de cliquer sur le bouton Compiler dans l'EDI. Faites-le pour chaque module modifié, car seul celui affiché par l'EDI est analysé.

Si vous utilisez des instructions complexes, ou des notations pointées en cascade, il devient difficile de trouver la raison de l'erreur d'exécution. Décomposez l'instruction en plusieurs instructions successives, et repérez celle qui part en erreur. Remontez ensuite en vérifiant chaque variable ou valeur utilisée.

Beaucoup d'erreurs sont dues à une faute d'orthographe. Relisez, relisez, passez à autre chose, revenez-y et relisez encore.

Respectez la casse des caractères (majuscule/minuscule) pour les arguments en chaîne de caractères, ainsi que pour les constantes API. Ce codage est un exemple typique :

```
DescrTri(0).Name = "SortFields"     ' Attention à la casse !
DescrTri(0).Value =  ConfigTri()
DescrTri(1).Name = "Orientation"    ' Attention à la casse !
' Constante API sur la ligne suivante, attention à la casse !
DescrTri(1).Value = com.sun.star.table.TableOrientation.ROWS
DescrTri(2).Name = "ContainsHeader" ' Attention à la casse !
DescrTri(2).Value = true
```

Erreur : variable indéfinie

Le message signifie plutôt : variable non définie. Il apparaît si vous avez utilisé l'option de déclaration obligatoire des variables :

```
Option Explicit
```

Quelque part sur la ligne en erreur, une variable n'a pas été déclarée préalablement. Souvent, il s'agit d'une simple faute de frappe. Relisez encore une fois... Et bénissez l'`Option Explicit`.

Erreur : variable objet non paramétrée

Premier cas

Vous utilisez une propriété ou une méthode d'une variable objet, mais cette variable est un objet vide (`Null`). Exemple :

```
Dim tata as object, toto as object
tata = toto.truc
```

L'erreur est que la variable `toto` n'est pas un véritable objet, puisque nous avons oublié de lui affecter un objet API. Basic ne peut pas trouver un `truc` dans cet objet !

Contre-exemple, ceci fonctionne sans erreur :

```
Dim tata as object, toto as object
tata = toto
```

En effet, cela revient à affecter à `tata` une variable objet de valeur `Null`.

En pratique, le cas arrive lorsqu'une fonction API est mal utilisée, mais qu'elle se contente de renvoyer un objet `Null`, sans erreur d'exécution :

```
Dim MonDocument As Object, lesFeuilles As Object
Dim AdresseDoc As String, PropFich()

AdresseDoc = "private:factory/calc"
MonDocument = StarDesktop.LoadComponentFromURL( _
               AdresseDoc,"_blank",0, PropFich)
lesFeuilles = MonDocument.Sheets
```

La dernière ligne déclenche une erreur parce que `MonDocument` est `Null`. En effet, il y a une erreur dans l'appel de `LoadComponentFromURL` qui l'empêche de créer un nouveau document Calc : on a affecté une valeur incorrecte à la variable `AdresseDoc`. Cherchez l'erreur !

Deuxième cas

Vous appelez une fonction de l'API ; un des arguments de cette fonction doit être un objet API, mais vous transmettez un autre type (valeur numérique, chaîne de caractères...), comme ici sur la dernière ligne :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursorByRange(1234)
```

Si vous n'utilisez pas `Option Explicit`, l'erreur peut aussi être une faute de frappe dans le nom d'une variable passée en argument à la fonction API : ce nom est alors interprété comme une nouvelle variable de type `Variant`. Si vous aviez utilisé l'option, le message d'erreur aurait été plus clair.

Erreur : utilisation incorrecte d'un objet

Cela signifie que vous essayez d'affecter à une variable objet quelque chose qui n'est pas un objet, par exemple une valeur numérique, ou une chaîne de caractères comme ici l'élément `Author` :

```
Dim tata as object
tata = ThisComponent.DocumentInfo.Author
```

Erreur : propriété ou méthode introuvable

Ce message est assez explicite. Vous cherchez à utiliser dans un objet (qui existe bien) une propriété ou une méthode que l'objet ne possède pas. Souvent, c'est une faute typographique, parfois c'est une mauvaise connaissance de l'API comme ici, pour un document Writer :

```
Dim tata as object
tata = thisComponent.DrawPages
```

La propriété est `DrawPage`, sans s...
... alors qu'un document Draw possède effectivement une propriété `DrawPages` !

Si l'erreur provient de ce que vous cherchez à deviner une propriété d'un objet API, vous perdez votre temps. Relisez plutôt ce livre, et utilisez la macro Xray.

Erreur : la sous-procédure ou procédure fonctionnelle n'est pas définie

Dans l'instruction en faute, Basic pense que vous essayez d'appeler une fonction ou procédure qu'il ne connaît pas. Vérifiez l'orthographe de la procédure que vous voulez utiliser, et vérifiez qu'elle existe bien. Exemple :

```
sleep(1000)
```

L'instruction ci-dessus donne cette erreur, car elle n'existe pas en Basic (mais elle existe dans d'autres langages). L'équivalent Basic est :

```
wait(1000)
```

Erreur : runtime exception

Traduction : anomalie à l'exécution. Sur un appel de fonction API, vous déclenchez un message d'erreur contenant :

```
Type: com.sun.star.uno.RuntimeException
Message:xxxxxx.
```

Parfois la section Type indique un nom d'exception plus spécifique. Bien souvent la section Message ne contient rien. C'est la fonction API qui vous renvoie un diagnostic suite à ses propres contrôles. Vérifiez chacun des arguments et relisez la documentation de l'API.

La raison est parfois plus subtile :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object, MonCadre As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor
MonCadre = MonDocument.createInstance("com.sun.star.text.TextFrame")
MonCadre.Width = 10400 ' 104 mm largeur
MonCadre.Height = 2530 ' 25,3 mm de haut
MonCadre.String = "hello"
```

Ci-dessus, il s'agit d'une erreur de méthodologie : l'API ne permet pas d'initialiser la propriété `String` du cadre avant de l'avoir inséré dans le texte.

Erreur : cannot coerce argument type during corereflection call !

Ce message en anglais est issu de l'API. Il n'est compréhensible que par un informaticien connaissant les mécanismes internes de OOoBasic. En termes courants, il signifie : vous m'avez transmis un argument dont le type n'est pas compatible avec celui que j'attends, je ne sais pas l'utiliser.

En pratique, un des arguments servant à appeler la méthode API est incorrect. À vous de trouver de quel argument il s'agit, et en quoi il est incorrect. Relisez la documentation de l'API ou le chapitre correspondant dans ce livre. Cet exemple essaie d'insérer un cadre dans un document Writer :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object, MonCadre As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor
MonTexte.insertTextContent( MonCurseur, MonCadre, false)
```

L'erreur est qu'on a oublié d'initialiser la variable `monCadre`.

L'API avec d'autres langages de programmation

Les autres langages de script de la version 2

Nous avons signalé au chapitre 3 les différents langages de scripts utilisables avec la version 2 d'OpenOffice.org et le niveau de leur intégration dans l'application. Voici le point de vue du programmeur.

JavaScript et BeanShell

Ces langages très similaires à Java nécessitent comme lui d'invoquer l'interface des méthodes à utiliser. Nous n'avons pas précisé les noms des interfaces dans ce livre car c'est parfaitement inutile avec Basic (et Python). La conséquence est une certaine lourdeur du codage, il est plus difficile à écrire et à relire. En contrepartie, les exemples du Developer's Guide étant écrits pour la plupart en Java sont facilement transposables.

Avec ces langages, la casse doit être respectée tant pour les noms de méthodes et de propriétés que pour tous les noms symboliques du programme. Les pseudo-propriétés de Basic n'existent pas, vous devez utiliser les méthodes de l'objet. Les pseudo-indexations de collections permises par OOoBasic doivent être remplacées par l'usage de la méthode `getByIndex()` de l'objet collection. Pour des interactions simples avec l'utilisateur, vous devrez développer des équivalents de `MsgBox` et `InputBox`.

Python

Dans Python, la casse doit être respectée tant pour les noms de méthodes et de propriétés que pour tous les noms symboliques du programme. Les pseudo-indexations de collections permises par OOoBasic doivent être remplacées par l'usage de la méthode `getByIndex()` de l'objet collection. Pour des interactions simples avec l'utilisateur, vous devrez développer des équivalents de `MsgBox` et `InputBox`.

Dans l'état actuel (pré-version 1.9 ou version 2.0.0), il est nécessaire de créer et éditer les scripts Python dans un éditeur externe produisant un codage source avec des fins de ligne façon Unix, même sous MS-Windows. Le fichier doit être mis dans un sous-répertoire `user/Scripts/python/` ou `share/Scripts/python`. Dans le premier cas, le sous-répertoire `python/` doit être créé par le développeur. Dans un document, le script doit être inséré en manipulant la structure du fichier document. Actuellement, le `sys.path` fourni ne permet d'importer que des modules situés dans `{installation}/program/` et ceux du `python-core`.

Tout ceci semble décourageant, pourtant les scripts obtenus gèrent l'API aussi facilement qu'en OOoBasic : pas d'invocation d'interface, utilisation possible des pseudo-propriétés. De plus, contrairement à Basic, il est possible avec Python de créer de véritables compo-

sants UNO intégrés à OpenOffice.org. Ils sont alors utilisés comme de nouveaux services, même dans une macro Basic.

Le programmeur profite des qualités de Python : l'indentation obligatoire facilite la relecture, les variables ne sont pas déclarées mais leur usage est contrôlé, elles peuvent changer de type dynamiquement, les algorithmes sont plus simples grâce aux fonctions puissantes intégrées dans Python et ses modules principaux, la gestion des erreurs est celle des langages modernes, et la programmation objet facilite la conception de programmes complexes.

Pour montrer la similarité de programmation API, voici le portage en Python de la portion de codage Basic indiqué plus haut, section « L'API réelle et l'API selon OOoBasic » :

```
mxDocCursor = mxDocText.createTextCursor()
mxDocCursor.gotoNextSentence(False)
mxDocCursor.gotoNextWord(False)
mxDocCursor.gotoNextWord(True)
mxDocText.insertString(mxDocCursor, "hello ", True)
mxDocCursor.CharWeight = uno.getConstantByName("com.sun.star.awt.FontWeight.BOLD")
```

Avant de vous lancer, lisez ces pages fondamentales en anglais sur le site udk :

- conversions entre types API et types Python, création de composant OpenOffice.org en Python : http://udk.openoffice.org/python/python-bridge.html ; cette page est aussi valide avec la version 1.1 d'OpenOffice.org ;
- mise en place de scripts Python et sorties de messages d'erreur dans OpenOffice.org 2.0 : http://udk.openoffice.org/python/scriptingframework/index.html

Piloter OpenOffice.org avec Microsoft COM

Sous MS-Windows, il est possible, à partir d'un langage de programmation indépendant, d'utiliser COM pour piloter l'application OpenOffice.org et manipuler des documents en utilisant son API telle qu'elle a été décrite dans cet ouvrage. Le SDK de la version 2.0 d'OpenOffice.org fournit plusieurs exemples :

- dans `examples/CLI/` : langages VB.NET et C#,
- dans `examples/OLE/` : langages Delphi et VbScript.

Dans la section CodeSnippets de OOoForum (http://www.oooforum.org/forum/) vous trouverez notamment le fil de discussion Using COM for OOo with different languages, qui présente des exemples de programmation COM avec divers langages dont VB, C#, Python, LotusScript, Delphi.

Mises à part l'initialisation de la connexion OLE et la syntaxe propre au langage, le cœur des instructions peut être remarquablement similaire à ce qui a été exposé. Voici l'équivalent en langage Delphi™ du codage exemple cité plus haut en Java et OOoBasic :

```
const
  _awtFontWeightBOLD = 150.000000;
var
  mxDocText, mxDocCursor : Variant;
// extrait du codage
  mxDocCursor:= mxDocText.createTextCursor;
  mxDocCursor.gotoNextSentence(false);
  mxDocCursor.gotoNextWord(false);
  mxDocCursor.gotoNextWord(true);
  mxDocText.insertString(mxDocCursor, 'hello ', true);
  mxDocCursor.CharWeight:= _awtFontWeightBOLD;
```

Comme nous n'avons pas un accès direct aux constantes nommées, nous devons définir celles dont nous avons besoin. Les valeurs de ces constantes sont lisibles dans les fichiers de l'arborescence `/idl` du SDK. Les constantes nommées « enum » sont implicitement numérotées à partir de zéro.

En Delphi, les noms des propriétés employés comme ci-dessus ne sont pas sensibles à la casse. Il en est de même pour les noms de méthodes. Les pseudo-propriétés sont aussi utilisables. Les tableaux réels, comme `ElementNames` fournis par certains objets, peuvent être indexés normalement. En revanche, les pseudo-indexations de collections permises par OOoBasic doivent être remplacées par l'usage de la méthode `getByIndex()` de l'objet collection.

La boîte à outils Delphi_OOo disponible en français sur le site fr.OpenOffice.org facilite la programmation OOo grâce à :

- une « unité » Delphi contenant des fonctions simplifiées d'accès à l'API,
- une « unité » listant toutes les constantes API et un programme pour la mettre à jour à partir d'un SDK,
- une « unité » apportant les fonctionnalités de Xray,
- des exemples de programmation,
- un mode d'emploi avec des conseils pour convertir une macro Basic en équivalent Delphi.

RESSOURCE

Le fil de discussion « Using COM for OOo with different languages » donne des exemples en Visual Basic, C, C++, C#, Python, Perl, Ruby, TCL et Delphi.

La documentation de l'API (Software Development Kit)

L'ensemble de la documentation est appelé SDK (Software Development Kit), conçu à l'origine pour un environnement de développement en Java ou en C++. Il se compose de diverses parties, d'où sa taille respectable :

- le « Developer's Guide », qui est un hypertexte expliquant la conception de l'API OpenOffice.org ;
- une version PDF imprimable du « Developer's Guide », plus de 1000 pages pour ceux qui veulent détruire les forêts, ou pour imprimer quelques pages d'un chapitre ;
- la référence IDL (Interface Definition Language) qui est un gigantesque hypertexte documentant (presque) tous les objets (au sens le plus général) de l'API ;
- les fichiers `*.idl` ayant servi à constituer la référence ; ce sont des fichiers texte où on peut lire les valeurs des constantes nommées ;
- une référence pour le développement en Java ;
- une référence pour le développement en C++ ;
- des exemples (repris dans le « Developer's Guide ») ;
- la spécification des formats XML utilisés ;
- divers outils de développement.

Le SDK est disponible pour consultation en ligne, à la page http://api.openoffice.org/ où vous trouverez aussi la dernière version à télécharger.
Il existe une version du SDK pour OOo 1.1.0 et une autre pour OOo 2.0. Dans cette dernière, vous trouverez aussi des informations concernant les versions 1.1.2 et suivantes, qui ne sont pas intégrées au SDK 1.1.0

Dans sa version hypertexte, le « Developer's Guide » contient de nombreux liens vers la référence IDL, et réciproquement, ce qui rend une lecture interactive souvent préférable à une lecture imprimée. D'ailleurs, la référence IDL est disponible seulement en version HTML.

La principale difficulté lorsqu'on étudie cette documentation est qu'elle est conçue pour un développeur Java ou C++ chargé de faire évoluer OpenOffice.org ou d'en réaliser une variante. Ce point de vue est intimement mélangé avec les descriptions des fonctionnalités disponibles, ce qui en rend la lecture assez difficile. D'autre part, et principalement dans le « Developer's Guide », les exemples sont donnés en langage Java, notablement plus lourd que OOoBasic.

La référence IDL est rédigée par les programmeurs eux-mêmes lors du développement, et compilée ensuite automatiquement. L'ennui, c'est que les programmeurs sont rarement intéressés par l'écriture de la documentation. Aussi est-elle parfois décevante par son aspect répétitif et ses lacunes.

Comment s'y retrouver ?

Nous vous conseillons d'installer le SDK sur votre ordinateur, pour un accès plus rapide, et d'utiliser un navigateur Internet capable d'afficher de multiples pages accessibles sous forme d'onglets, par exemple Mozilla Firefox (http://frenchmozilla.org/).

Dans une installation standard sous MS-Windows, la page d'entrée se trouve sur votre disque à l'adresse `C:\OpenOffice.org1.1_SDK\index.html`. En fait, cette page est assez peu utilisée. Vous commencerez plutôt par la page « Developer's Guide » ou la page « IDL Reference », qui sont accessibles depuis la première.

La page d'introduction au « Developer's Guide » vous renvoie vers le sommaire (*Table of contents*) de ce qui est en fait un véritable livre. Beaucoup de recherches générales partiront de ce sommaire. Comme certaines pages sont très lentes à charger sur un navigateur, il peut être plus pratique d'afficher la version PDF du « Developer's Guide ».

La première page de la référence IDL liste les différents modules qui composent le module `Star`, l'ensemble de l'application. Tout est ensuite décomposé en un arbre avec de nombreuses ramifications. La position d'une page dans cet arbre est rappelée en haut à gauche des pages, par exemple `::com::sun::star::`, et l'organisation des répertoires contenant les pages HTML reflète exactement cette hiérarchie. Lorsque nous indiquons une référence de documentation comme `com.sun.star.drawing.LineProperties`, vous afficherez la page correspondante en suivant, depuis la première page de l'IDL, le lien intitulé `drawing`, puis dans la page obtenue le lien intitulé `LineProperties`, comme on peut le voir sur la figure A-1.

L'autre moyen d'accès à l'IDL, à partir de n'importe laquelle de ses pages, est d'utiliser le lien Index, sur la première ligne du haut de la page. Il vous affiche la page `A` d'un dictionnaire. Si vous cherchez la documentation sur `LineProperties`, affichez la page `L` et recherchez ce mot dans celle-ci, en début de ligne. Vous repérerez la ligne :

```
LineProperties - service ::com::sun::star::drawing:: .LineProperties
```

Ici, il s'agit d'un service dont la page de description est accessible par lien hypertexte. Dans bien des cas, vous obtiendrez plusieurs lignes avec le même nom, car plusieurs types d'objets ayant une propriété ou une méthode similaire ont normalement le même nom. Ce sera à vous de déterminer lequel correspond à votre contexte, grâce aux autres informations de la ligne.

Un dernier moyen, très rapide, d'accéder directement à la bonne page de l'IDL consiste à utiliser l'outil XRay, dont nous avons déjà parlé.



Figure A–1
Une page de la référence IDL

Conclusion

Une compréhension de l'API elle-même hors du contexte des macros et une connaissance des sources d'information disponibles nous semblent nécessaires pour parvenir à en tirer parti.

L'annexe suivante offre un panorama de routines utilitaires, qui complète utilement les techniques vues au chapitre 19.

B

Routines utilitaires

Les sous-programmes présentés dans cette annexe seront souvent utilisés dans vos projets de macros. Certains sont soumis à la licence LGPL, décrite sur le site http://www.gnu.org/copyleft/lesser.html.

LICENCE LGPL

Une traduction non officielle de la licence LGPL est disponible sur le site http://www.linux-france.org/article/these/licence/lgpl/lgpl_monoblock.html. Cette licence stipule que vous pouvez utiliser ces routines dans vos programmes et les améliorer, à condition de maintenir les indications de licence et d'auteur initial.

La première partie des routines décrites ici ont déjà été utilisées dans divers chapitres de ce livre. Elles peuvent servir de base à des constructions plus élaborées, notamment sur le plan du traitement d'erreurs.

Tableaux de propriétés

Certains objets de l'API se présentent sous la forme d'un tableau de propriétés dont la liste est prédéfinie. C'est le cas des descripteurs de tri pour Calc et Writer.

La routine `printProps` affiche les noms de ces propriétés et leur rang dans le tableau. La fonction `getPropVal` renvoie la valeur de la propriété dont le nom est en argument. La routine `setPropVal` affecte une valeur à une propriété.

```
rem CodeAnnexeB-01.sxw   bibli : Proprietes Module1
Option Explicit

' affiche les noms de propriétés du tableau descr
Sub printProps(descr As Variant)
Const Titre = "Tableau de propriétés"
Dim x As Long, liste As String
for x = 0 to UBound(descr)
  liste = liste & "Index " & x & " : " _
                &  descr(x).Name & chr(13)
next
MsgBox(liste, 0, Titre)
End Sub


' affecte la valeur valProp à la propriété nomProp
Sub setPropVal(descr As Variant, _
               nomProp As String, valProp As Variant)
Const Titre = "Tableau de propriétés"
Dim x As Long
for x = 0 to UBound(descr)
  if descr(x).Name = nomProp then
    descr(x).Value = valProp
    Exit Sub
  end if
next
MsgBox("Propriété inconnue : " & nomProp, 16, Titre)
End Sub


' renvoie la valeur de la propriété nomProp
Function getPropVal(descr As Variant, nomProp As String)_
                    As Variant
Const Titre = "Tableau de propriétés"
Dim x As Long
for x = 0 to UBound(descr)
  if descr(x).Name = nomProp then
    getPropVal = descr(x).Value
    Exit Function
  end if
next
MsgBox("Propriété inconnue : " & nomProp, 16, Titre)
End Function
```

Collections

Certaines collections d'objets ne permettent pas un accès direct par le nom d'objet. La fonction `getIndexByName` recherche dans la collection l'objet dont le nom est passé en argument. En cas de succès, elle renvoie l'index correspondant ; en cas d'échec, elle affiche une erreur et renvoie un index hors-limites.

```
rem CodeAnnexeB-01.sxw   bibli : Collections Module1
Option Explicit

' renvoie l'index de l'élément ayant le nom donné en argument
Function getIndexByName(collection As Object, _
                        leNom As String) As Long
Const Titre = "Collection"
Dim x As Long
for x = 0 to collection.Count -1
  if collection(x).Name = leNom then
    getIndexByName = x ' renvoyer l'index correspondant au nom
    Exit Function
  end if
next
MsgBox("Nom inconnu : " & leNom, 16, Titre)
getIndexByName = -100 ' valeur d'index hors-limites
End Function
```

Coordonnées de cellules

L'API fournit des fonctions pour convertir en adresse numérique une « adresse utilisateur » de cellule ou de zone de cellules. Les fonctions de cette bibliothèque réalisent l'inverse, sur toute l'étendue des adresses possibles. La version 2 d'OpenOffice.org devrait repousser les limites de Calc au-delà de 32 000 lignes. Ces macros peuvent facilement être adaptées, mais plusieurs fonctions Basic pourront aussi être modifiées par cette évolution.

La fonction `adrZoneString` renvoie la chaîne de caractères correspondant aux coordonnées de la zone de cellules passée en argument. La fonction `adresseString` effectue le même travail pour une adresse de cellule. Les autres fonctions sont utilisées par les deux précédentes.

Notez que ces routines effectuent un contrôle de vraisemblance sur les coordonnées fournies. En cas d'anomalie, la chaîne renvoyée comportera des caractères « ?? ».

```
rem CodeAnnexeB-01.sxw   bibli : Cellules Module1
Option Explicit

' convertit une adresse de zone de cellules en adresse textuelle
Function adrZoneString(maDoc As Object, adrZone As Object) As String
Dim resu As String
resu = maDoc.Sheets(adrZone.Sheet).Name & "." & _
               alphaXY(adrZone.StartColumn, adrZone.StartRow)
if (adrZone.StartColumn <> adrZone.EndColumn) or _
   (adrZone.StartRow <> adrZone.EndRow) then
  resu = resu & ":" & alphaXY(adrZone.EndColumn, adrZone.EndRow)
end if
adrZoneString = resu
End Function


' convertit une adresse de cellule en adresse textuelle
Function adresseString(maDoc As Object, adrCellule As Object) As String
adresseString = maDoc.Sheets(adrCellule.Sheet).Name & "." & _
               alphaXY(adrCellule.Column, adrCellule.Row)
End Function

' convertit une coordonnée XY en coordonnée alphanumérique
Function alphaXY(X As Long, Y As Long) As String
if (Y>=0) and (Y<32000) then
  alphaXY = lettreColonne(X) & CStr(Y +1)
else
  alphaXY = lettreColonne(X) & "??"
end if
End Function


' convertit numéro de colonne 0...255 en lettres A...IV
Function lettreColonne(n As Long) As String
Select Case n
Case > 255
  lettreColonne = "??"
Case < 0
  lettreColonne = "??"
Case < 26
  lettreColonne = _Lettre(n)
Case Else
  lettreColonne = _Lettre((n \ 26) -1) & _Lettre(n Mod 26)
End Select
End Function

' fonction interne
Function _Lettre(p As Long) As String
_Lettre = chr(Asc("A") +p)
End Function
```

Rechercher un objet par son nom

La fonction `findObjectByName` est inspirée des travaux de Danny Brewer. Cette fonction recherche dans une page de dessin un objet dont le nom est donné en argument. En cas d'échec, la fonction renvoie la valeur `Null`.

```
rem CodeAnnexeB-02.sxw   bibli : Dessin Module1
Option Explicit

' retrouve un objet à partir de son nom
Function FindObjectByName(unePage As Object, _
         nomObj As String, Optional service As String) As Object
Dim objX As Object, x As Long
  For x = 0 To unePage.Count - 1
    objX = unePage(x)
    If objX.Name = nomObj Then
      if IsMissing(service) then
        FindObjectByName = objX ' objet trouvé
        Exit Function
      else
        if objX.supportsService(service) then
          FindObjectByName = objX ' objet trouvé
          Exit Function
        end if
      end if
    EndIf
  Next
End Function ' renvoie Null en cas d'échec
```

Le paramètre `service` est optionnel. Il permet de ne rechercher qu'un objet proposant le service indiqué. En effet, une page de dessin peut contenir différentes sortes d'objets, et vous pourriez par exemple obtenir une image ayant le nom du dessin que vous cherchez (par programmation, on peut donner le même nom à plusieurs objets d'une page). En vérifiant que l'objet obtenu reconnaît un service caractéristique du type d'objet recherché, nous effectuons une vérification supplémentaire. Le tableau B-1 indique quel service caractérise un objet.

Tableau B–1 Services caractéristiques

Type d'objet recherché	Service caractéristique
Dessin, sauf 3D	`com.sun.star.drawing.LineProperties`
Dessin 3D	`com.sun.star.drawing.Shape3DScene`
Image	`com.sun.star.drawing.GraphicObjectShape`
Objet OLE2	`com.sun.star.drawing.OLE2Shape`

Par exemple, pour rechercher seulement une forme dessinée appelée « F3 », nous écrirons :

```
Dim sv As String
sv = "com.sun.star.drawing.LineProperties"
maForme = FindObjectByName(maPage, "F3", sv)
```

En effet, le service `Shape`, trop général, est également proposé par une image. En revanche, vous pouvez être plus précis et exiger par exemple une `EllipseShape`.

Rechercher un diagramme par son nom

La fonction `FindChartByObjName` recherche dans une feuille Calc un objet diagramme à partir du nom de forme défini par l'interface utilisateur. Elle utilise la fonction précédente, en recherchant un objet `OLE2`. Toutefois, pour être certain qu'il s'agit d'un diagramme, elle vérifie que le sous-objet `Model` propose le service de diagramme. La fonction renvoie la valeur `Null` en cas d'échec.

```
rem CodeAnnexeB-02.sxw   bibli : Dessin Module1
Option Explicit

' retrouve un objet diagramme à partir du nom de l'objet
Function FindChartByObjName(laFeuille As Object, _
                            nomDiag As String) As Object
Dim dessin As Object, sv As String

sv = "com.sun.star.drawing.OLE2Shape"
dessin = FindObjectByName(laFeuille.Drawpage, nomDiag, sv)
if not IsNull(dessin) then
  if dessin.Model.supportsService(_
              "com.sun.star.chart.ChartDocument") then
    FindChartByObjName = dessin.Model
  end if
end if
End Function ' renvoie Null en cas d'échec
```

Redimensionner une image

Le sous-programme `resizeImageByWidth` redimensionne une image à une largeur donnée, en gardant ses proportions. Il obtient de l'objet image l'objet `GraphicObjectFillBitmap`, qui donne accès aux informations sur l'image elle-même. Dans ce dernier objet, la structure `Size` nous donne la taille de l'image mesurée en pixels. Une règle de trois nous permet d'en déduire les dimensions en 1/100 de millimètres.

```
rem CodeAnnexeB-02.sxw   bibli : Images Module1
Option Explicit

Sub resizeImageByWidth(uneImage As Object, largeur As Long)
Dim leBitMap As Object, Proportion As Double
Dim Taille1 As New com.sun.star.awt.Size

LeBitmap = uneImage.GraphicObjectFillBitmap
Taille1 = LeBitMap.Size ' taille en pixels !
Proportion = Taille1.Height / Taille1.Width
Taille1.Width = largeur ' largeur en 1/100 de mm
Taille1.Height = Taille1.Width * Proportion
uneImage.Size = Taille1
End Sub
```

Vous pourrez facilement réaliser sur ce modèle un sous-programme redimensionnant selon une hauteur donnée.

Autre variation, le sous-programme `resizeImageByDPI` redimensionne l'image en respectant une densité de points, donnée en DPI.

```
rem CodeAnnexeB-02.sxw   bibli : Images Module2
Option Explicit

Sub resizeImageByDPI(uneImage As Object, DPI As Long)
Dim leBitMap As Object, Proportion As Double
Dim Taille1 As New com.sun.star.awt.Size
Const pouce = 2540 ' longueur en 1/100 de mm

LeBitmap = uneImage.GraphicObjectFillBitmap
Taille1 = LeBitMap.Size ' taille en pixels !
Proportion = pouce / DPI
Taille1.Width =  Taille1.Width * Proportion
Taille1.Height = Taille1.Height * Proportion
uneImage.Size = Taille1
End Sub
```

Traduire un nom de style

Le nom d'un style obtenu par exemple avec la propriété `ParaStyleName` est le nom interne, en anglais. Le nom affiché dans le styliste est dans la langue locale.

La fonction `getLocaleStyleName` renvoie le nom localisé correspondant à un nom interne. Elle emploie trois arguments :

- l'objet document (qui contient les styles),
- le nom de la famille de styles,
- le nom anglais du style.

Cette fonction ne donnera pas de bons résultats dans Impress pour les styles de la famille `Standard` (voir le chapitre 13).

```
rem CodeAnnexeB-03.sxw   bibli : NomsStyles Module1
Option Explicit

' renvoie le nom localisé d'un style
Function getLocaleStyleName(leDoc As Object, _
                 fam As String, nomStyle As String) As String
Dim uneFamille As Variant
Dim desStyles As Object, unStyle As Object

on Error Goto pbStyle
desStyles = leDoc.StyleFamilies.getByName(fam)
unStyle = desStyles.getByName(nomStyle)
getLocaleStyleName = unStyle.DisplayName
On Error Goto 0
exit function

pbStyle:
  On Error Goto 0
  getLocaleStyleName = "????"
End Function
```

Nous avons utilisé un traitement d'erreur pour renvoyer des points d'interrogation sur les cas d'échec, notamment si le nom de famille de styles ou le nom de style est inconnu.

Le document comporte un exemple de routine utilisant la fonction. Vous remarquerez que si vous entrez un nom localisé, la fonction renvoie ce même nom, grâce à la souplesse de `getByName`.

Rappel des routines déjà décrites dans les chapitres

Dialogue

La fonction `Dialogue` crée un objet dialogue connaissant le nom de la boîte de dialogue et sa bibliothèque. Le sous-programme `CenterDialog` centre un nouveau dialogue par rapport à un dialogue père. Voir le chapitre 16 pour ces deux routines.

Tableur et base de données

Les fonctions suivantes se trouvent dans le fichier `CalcSQL.sxc` qui se trouve dans le répertoire regroupant les exemples du chapitre 17.

La fonction `CALCSQL1` sert à effectuer dans une cellule Calc une requête SQL sur une base de données. Elle renvoie un tableau de résultats.

La fonction `CALCSQL2` importe le résultat d'une requête SQL depuis une base de données. Le résultat est obtenu dans un tableau de cellules d'une feuille du tableur.

Rechercher la forme d'un contrôle de formulaire

La fonction `FindCtrlShapeByName` recherche dans une page de dessin une forme correspondant à un contrôle de formulaire dont le nom est donné en argument. Voir le chapitre 18.

Création et décompression d'un fichier ZIP

Plusieurs sous-programmes de gestion de fichier ZIP sont décrits dans le chapitre 19.

Les routines suivantes n'ont pas été utilisées dans les chapitres précédents, mais elles sont d'une incontestable utilité.

Conversion date-heure vers heure, minute, seconde

Les fonctions Basic `Hour`, `Minute`, `Second` sont incorrectes sur les versions d'OpenOffice.org antérieures à 1.1.4, et corrigées à partir de celle-ci et sur la version 2.0.0. Pour les utilisateurs des versions posant problème, nous proposons le sous-programme `Time_HMS` qui réalise une conversion correcte. Il a été testé sur les 86 400 valeurs possibles de la partie heure.

Le sous-programme `Time_HMS` reçoit une valeur de date-heure au format interne, et en extraie les valeurs d'heure, minute, seconde.

- Argument 1 : date-heure
- Argument 2 : variable qui recevra la valeur d'heure (0 à 23)

- Argument 3 : variable qui recevra la valeur de minute (0 à 59)
- Argument 4 : variable qui recevra la valeur de seconde (0 à 59)

Codage du sous-programme :

```
rem CodeAnnexeB-04.sxw   bibli : Standard Module1
Option Explicit

'Convertit une heure de type Date en heure, minute, seconde
Sub Time_HMS(temps As Date, _
         hr As Integer, mn As Integer, sec As Integer)
Dim s2 As Long, m2 As Long, h2 As Long
Dim d3 As Double, s3 As Long

d3 = CDbl(temps) ' conversion en Double
d3 = d3 - Fix(d3) ' garder la partie fractionnaire
s3 = CLng(d3 * 86400) ' convertir le temps en secondes
' le calcul se poursuit avec des entiers Long
s2 = s3 Mod 60
s3 = (s3 -s2) / 60
m2 = s3 Mod 60
h2 = (s3 -m2) / 60
hr = h2
mn = m2
sec = s2
End Sub
```

Exemple d'utilisation :

```
rem CodeAnnexeB-04.sxw   bibli : Standard Module2
Option Explicit

Sub Exemple_Time_HMS()
Dim t1 As Date
Dim h1 As Integer, m1 As Integer, s1 As Integer

t1 = Now
Time_HMS(t1, h1, m1, s1)
MsgBox(CStr(t1) & chr(13) & _
       "    " & h1 & ":" & m1 & ":" & s1)
End Sub
```

Traitement des chaînes de caractères longues

Sur les versions 1.1 et antérieures d'OpenOffice.org, les instructions OOoBasic `InStr`, `Left`, `Right` et `Asc` ne traitent pas de manière correcte les chaînes de caractères de plus de

32 767 caractères. Elles sont corrigées à partir de la version 2.0.0. Pour les utilisateurs des versions posant problème, le document `CodeAnnexeB-07.sxw` contient dans la bibliothèque `LongString` des fonctions de remplacement : `InStrL`, `LeftL`, `RightL` et `AscL`. Elles utilisent les mêmes arguments que les fonctions d'origine.

Remplacer un motif partout dans une chaîne de caractères

La fonction `RemplaceChaine`, disponible sur la page Internet http://fr.openoffice.org/index.html a pour but de remplacer une séquence de caractères par une autre, chaque fois qu'elle apparaît dans une chaîne de caractères. Un exemple trivial consiste à remplacer les caractères de séparation dans un chemin de fichier. Les arguments de la macro sont, dans l'ordre :

- la chaîne initiale,
- la séquence à rechercher (chaîne de caractères),
- la séquence qui doit la remplacer (chaîne de caractères),
- un indicateur mis à `True` pour que la recherche distingue la casse,

La fonction renvoie une chaîne de caractères modifiée.

Cette macro est codée afin de fonctionner aussi sur des chaînes de plus de 32 767 caractères.

Trier un tableau de données

La fonction `TriShell`, disponible sur la page Internet http://fr.openoffice.org/index.html effectue le tri d'un tableau (`Array`) de `Variant`. Elle est facilement modifiable pour trier une structure quelconque.

Le cœur de la macro est la comparaison des éléments et leur échange. Adaptez-le à vos besoins, par exemple une comparaison de `String` tenant compte de la casse.

Ainsi que le précise le texte du document contenant la macro, des théoriciens ont étudié avant nous comment optimiser un tri, alors c'est l'occasion de ne pas réinventer la roue (en plus mal). La méthode *Shell* est très efficace sans nécessiter un codage récursif. Elle est probablement assez rapide pour vos besoins en Basic, et pas plus compliquée à employer qu'un tri *Bulle*, méthode évidente mais très lente.

Obtenir des adresses URL de fichiers et répertoires

Le document `CodeAnnexeB-05.sxw` contient plusieurs fonctions facilitant l'analyse d'adresses de chemins de fichiers. Elles sont écrites en anglais pour être comprises du plus grand nombre.

Toutes ces fonctions (voir le tableau B-2) utilisent des adresses au format URL. Toutes les adresses de répertoires doivent se terminer par le caractère `/`.

Tableau B–2 Fonctions d'adresses URL

Fonction	Argument	Résultat
`getDirectory`	Chemin d'un fichier	Chemin du répertoire contenant le fichier
`getParentDir`	Chemin d'un répertoire	Chemin du répertoire parent
`getFullFileName`	Chemin d'un fichier	Nom et extension du fichier, exemple : `monfichier.sxw`
`getFileNameOnly`	Chemin d'un fichier	Nom du fichier sans l'extension, exemple : `monfichier`
`getFileExt`	Chemin d'un fichier	Extension du fichier, avec le point, exemple : `.sxw`

Copier-coller avec le presse-papiers

C'est avec un peu de scrupules que nous présentons ces deux macros, qui ont été réalisées avec l'enregistreur de macros ; c'est cependant une solution simple et efficace et nous en avons amélioré le code obtenu.

```
rem CodeAnnexeB-06.sxw   bibli : PressePapier Module1
Option Explicit

Sub CopierDansPressePapier()
dim fenetreDoc as object, dsp as object
fenetreDoc = StarDesktop.CurrentFrame
dsp = createUnoService("com.sun.star.frame.DispatchHelper")
dsp.executeDispatch(fenetreDoc, ".uno:Copy", "", 0, Array())
End Sub

Sub CollerDepuisPressePapier()
dim fenetreDoc as object, dsp as object
fenetreDoc = StarDesktop.CurrentFrame
dsp = createUnoService("com.sun.star.frame.DispatchHelper")
dsp.executeDispatch(fenetreDoc, ".uno:Paste", "", 0, Array())
End Sub
```

Les deux macros doivent être lancées sans utiliser le bouton de l'EDI. En premier plan doit s'afficher une fenêtre gérée par OpenOffice.org, en général celle d'un document ouvert. Notez que même la fenêtre de l'EDI conviendra !

La première macro copie dans le presse-papiers la zone sélectionnée par l'utilisateur (ou sélectionnée par macro puis rendue visible pour l'utilisateur, comme expliqué dans la partie Writer). Pour cela, elle récupère la fenêtre courante affichée par OpenOffice.org (alors que le codage de l'enregistreur de macros récupère la fenêtre courante du document). L'utilisation du service `DispatchHelper` est reprise du code de l'enregistreur de macros, le nom de la variable ayant été changé pour raison de mise en pages.

La deuxième macro copie le presse-papiers dans la fenêtre OpenOffice.org en premier-plan, en écrasant éventuellement une zone sélectionnée. L'utilisateur peut afficher une autre application OpenOffice.org, alors que le codage original ne le permettrait pas. La seule différence avec la première macro réside dans le deuxième argument de la méthode `executeDispatch`.

À NOTER L'enregistreur de macros

Utiliser l'enregistreur de macros est quelquefois frustrant, mais cela peut être un moyen simple de couvrir une fonctionnalité difficile ou impossible à implémenter avec l'API.

La liste des commandes du dispatch est disponible sur internet aux adresses suivantes :

- pour la version 1.1 d'OpenOffice.org :
  - http://framework.openoffice.org/files/documents/25/1042/commands_11beta.html
- pour la version 2.0 d'OpenOffice.org :
  - http://framework.openoffice.org/files/documents/25/2570/commandsReference.html

Les commandes de la forme .uno:xxxxx sont dans la première colonne des tableaux. L'utilisation des constantes numériques (ID) de cette page est à proscrire. Elles sont à usage interne et peuvent changer à tout moment.

Conclusion

Les macros exposées dans ce chapitre sont directement opérationnelles. Elles illustrent la notion de « ré-utilisabilité » qui permet de gagner du temps et de l'énergie en pérennisant les développements. Il est inutile de ré-écrire plusieurs fois le même code pour effectuer la même action. Usez et abusez de telles routines ; elles constitueront progressivement une boîte à outils adaptée à vos besoins, et vous permettront de rester concentré sur l'objectif premier de la macro que vous êtes en train de concevoir.

Voyons maintenant quelles richesses sont disponibles sur l'Internet.

C

Ressources disponibles sur Internet

De nombreuses informations, outils et compléments sur la programmation OpenOffice.org sont disponibles sur Internet. Ils sont une source toujours renouvelée de savoir et d'inspiration.

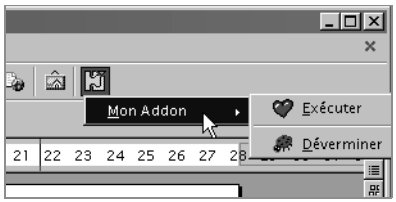
> **ATTENTION Lecture critique**
>
> De nombreuses macros sont disponibles sur Internet, mais elles ne doivent pas être considérées comme une référence absolue. Elles peuvent contenir des erreurs, ou être améliorées, ou correspondre aux premiers essais d'un programmeur ou à des extraits d'un codage plus complexe. Cherchez donc toujours à comprendre les principes utilisés.

Diffuser vos macros avec un add-on

Installer des macros sur un système et ajouter des icônes pour les déclencher est une tâche intimidante pour un utilisateur ordinaire. Pour un utilisateur confirmé, répéter ces opérations sur plusieurs ordinateurs devient lassant et sujet à erreurs. La méthode des *add-on* est prévue pour modifier la configuration d'une installation OpenOffice.org. En particulier, elle permet d'installer une bibliothèque de macros, d'ajouter des sous-menus ou des boutons avec icônes (à des endroits réservés pour cet usage) et même de prévoir des

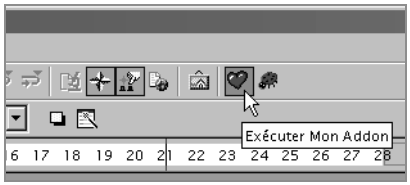
libellés en différentes langues. La figure C-1 montre un sous-menu créé pour accéder à l'*add-on*.

Figure C–1
Sous-menu pour un add-on



La figure C-2 montre deux boutons ajoutés sur la barre d'outils de la version 1.1 d'OpenOffice.org. Dans la version 2.0, chaque *add-on* est accessible dans une barre d'outil supplémentaire qui peut être juxtaposée aux autres.

Figure C–2
Nouveaux boutons pour un add-on



Ceci est expliqué dans le document « *HowTo* - Comment diffuser vos macros avec un *Add-on* ». Le document est lui-même un outil (réalisé par macros) qui sert à créer l'*add-on* sous forme d'un fichier ZIP.

- http://fr.openoffice.org/Documentation/How-to/indexht.html, section programmation Basic, document n°4.

Avec la version 1.1 d'OpenOffice.org, le fichier *add-on* doit être installé en exécutant le programme `pkgchk` à partir de la ligne de commande. La méthode à suivre est indiquée dans le document, mais certains utilisateurs ont quelques difficultés à le faire correctement, aussi d'autres développeurs ont-ils cherché une méthode plus simple pour diffuser un *add-on*. Parmi ceux-ci, Didier Lachièze a réalisé le « *HowTo* - Installeur de macros ». D'autres travaux sont en cours.

- http://fr.openoffice.org/Documentation/How-to/indexht.html, section programmation Basic, document n°6.

Avec la version 2.0 d'OpenOffice.org, l'installation d'un *add-on* est beaucoup plus simple, car on peut utiliser le Gestionnaire de packages, qui est accessible à partir du menu Outils. La figure C-3 montre un *add-on* déjà installé.

Contrairement à la version 1.1 d'OpenOffice.org, un add-on peut être ajouté dynamiquement à OpenOffice.org même si celui-ci est en cours d'execution : cliquer sur le bouton Ajouter et choisir le fichier `.zip` de l'add-on. Le gestionnaire de packages peut aussi être exécuté en ligne de commande par l'intermédiaire de la commande `unopkg`.

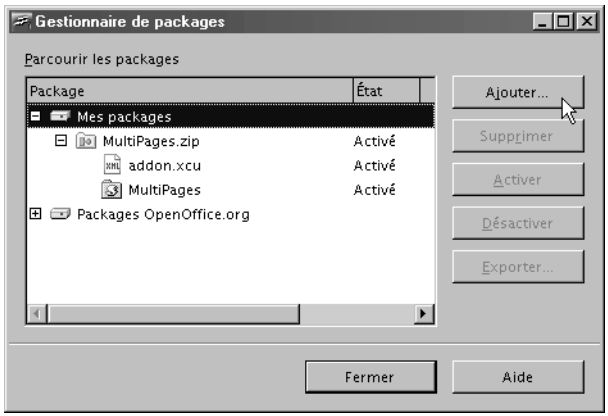


Figure C–3
Le Gestionnaire de packages

Introspection et documentation avec Xray

L'API offre des fonctions d'introspection et de core reflection permettant d'obtenir à l'exécution de nombreuses informations sur les objets manipulés. Toutefois, ces fonctions sont assez complexes à utiliser.

L'outil Xray, réalisé avec des macros, met en forme ces informations et permet d'étudier les sous-objets. Il est capable de retrouver dans l'IDL la documentation du sous-objet, si elle existe. Les auteurs utilisent intensivement Xray pour étudier la structure des objets et lire leur documentation dans l'API. Sans Xray, cet ouvrage ne serait pas aussi détaillé.

- http://fr.openoffice.org/Documentation/How-to/indexht.html
  section programmation Basic, document n°5.

Xray se présente sous la forme de deux bibliothèques de macros à installer dans le conteneur `soffice` (dans Mes Macros pour la version 2.0 d'OpenOffice.org). Pour étudier un objet, il est nécessaire d'ajouter une instruction dans le codage, pour appeler Xray :

```
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Janvier")
xray.xray maFeuille
```

L'expression signifie : appeler la macro `xray` qui se trouve dans la bibliothèque `xray` et lui transmettre en argument l'objet `maFeuille`. On obtient le panneau de la figure C-4, où on visualise les propriétés disponibles pour l'objet analysé. En positionnant le curseur sur la ligne d'un élément complexe comme `CellProtection`, il suffit de cliquer le bouton Xray pour afficher le contenu de ce sous-objet. L'opération peut être répétée sur différents sous-objets, à plusieurs niveaux.

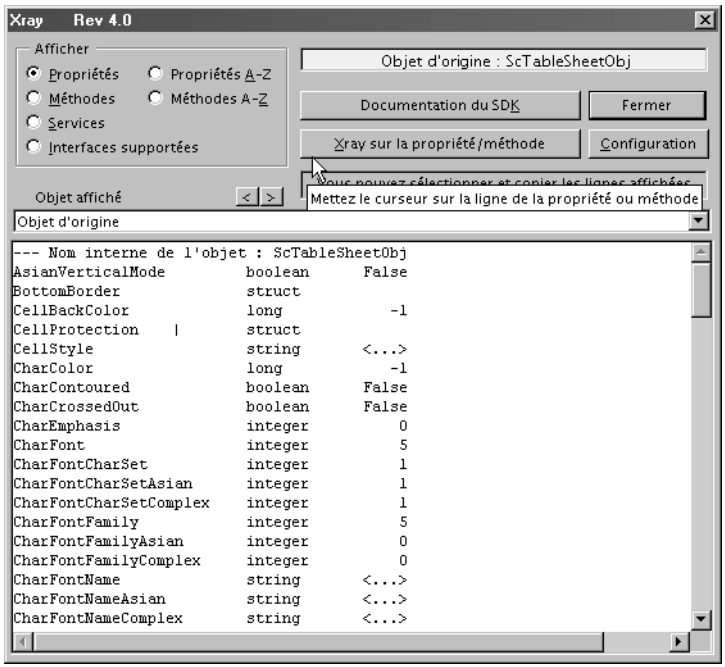


Figure C–4
Xray : propriétés d'une feuille Calc

En positionnant le curseur sur la ligne d'un élément, un simple clic sur le bouton `Documentation` permet de visualiser sur votre navigateur Internet la documentation API concernant l'objet sélectionné (voir figure C-5). Pour utiliser cette fonction, il faut toutefois avoir installé la documentation SDK sur l'ordinateur.

La figure C-6 montre les méthodes offertes par l'objet en analyse. On peut approfondir l'analyse sur une des méthodes à condition qu'elle ne comporte aucun argument ou encore accéder à sa documentation.

Xray liste les services proposés par l'objet et ceux disponibles par invocation, ou liste les interfaces prises en charge. Là encore, la documentation d'un service ou d'une interface est directement accessible.

Une version équivalente de Xray pour le langage Python est disponible. Naturellement appellée pyXray (voir figure C-7), elle a été réalisée par Laurent Godard et est disponible sur le site http://www.indesko.com/ . Cet outil est utilisable dès la version 1.1 d'OpenOffice.org.

Il faut une certaine expérience de l'API pour ne pas être perdu dans les informations fournies. Comme les objets API héritent pour la plupart des propriétés et méthodes d'un ou plusieurs autre(s) objet(s), et ainsi de suite, et que vous pouvez analyser un objet interne à un objet, vous retrouvez certaines méthodes et propriétés des « briques de base » d'OpenOffice.org, par exemple `setDelegator`, `getByName`, ou le tableau `ImplementationId`.

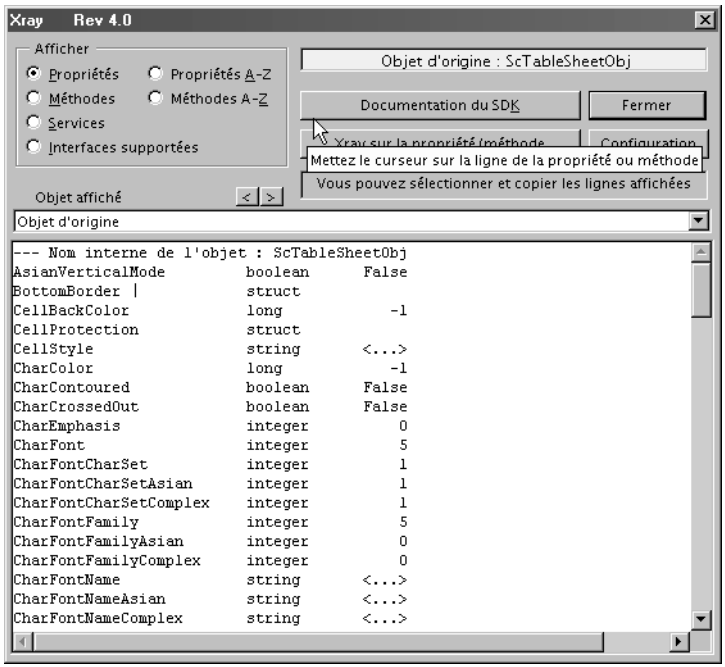


Figure C–5
Xray : voir la documentation d'un objet

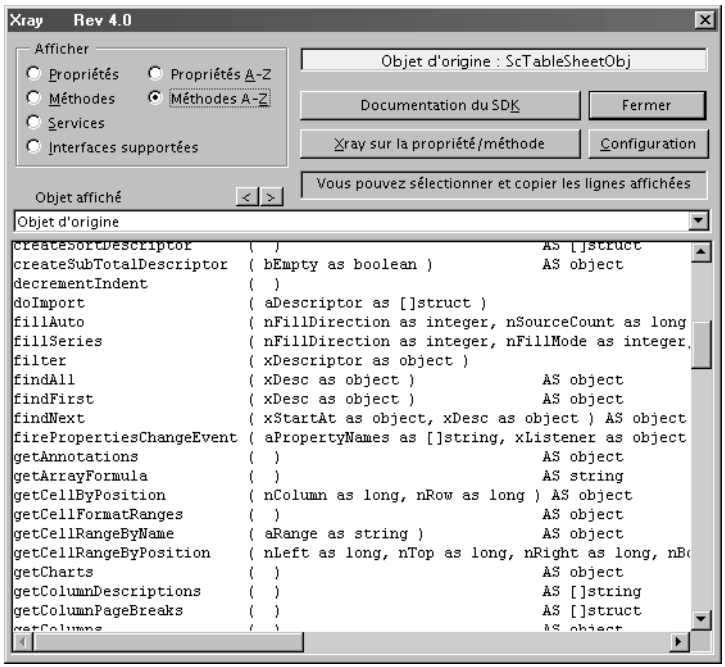


Figure C–6
Xray : méthodes d'une feuille Calc

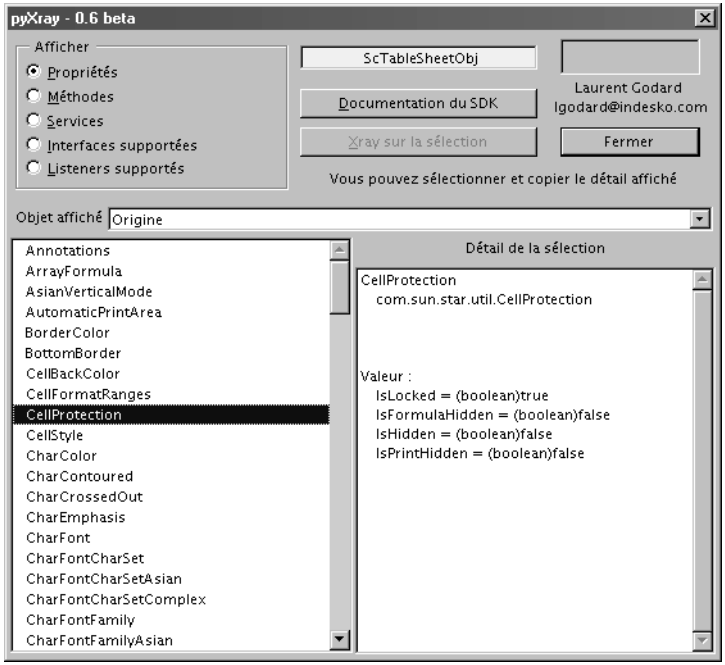


Figure C–7
Xray, version Python

Certaines propriétés d'objet sont listées mais non disponibles dans le contexte, d'autres renvoient un objet qui n'est autre que l'objet lui-même.

Lorsque vous demandez la documentation sur un élément d'un objet, la recherche peut échouer : la documentation manque, ou l'élément est un « fossile » des temps révolus, ou l'objet n'est pas officiellement utilisable.

Les bibliothèques de DannyB

DannyB est la signature de Danny Brewer. Il propose, sur la page Developers du site OOoMacros, des exemples et des routines réutilisables, dont :

- *Danny Brewer's Examples* (basé sur les réponses données dans OOoForum),
- *Danny's Draw Power Tools* (outils de macros pour Draw),
- *Danny's Library* (bibliothèque de l'auteur).

Il a publié aussi dans la section Code Snippets de OOoForum plusieurs bibliothèques personnelles, mises à jour au long de ses recherches ; notamment :

- Danny's Basic library
- Danny's Python Modules

- String utility functions for OOo Basic

▸ http://ooomacros.org/dev.php
Comme vous pouvez vous en douter, Danny Brewer écrit en anglais...

Les ressources que nous venons de passer en revue sont essentiellement destinées au développeur. Nous allons voir quelles sont celles mises à la disposition des utilisateurs.

Documents informatifs et exemples

La page des HowTo du site français d'OpenOffice.org contient une section intéressante sur la programmation Basic.

▸ http://fr.openoffice.org/Documentation/How-to/indexht.html

Une partie de ces « HowTo » fait double emploi avec notre ouvrage ou sont cités ailleurs, mais nous signalerons ceux apportant des informations complémentaires.

- « Dans la jungle de l'API » : une expérience vécue d'un essai de compréhension de l'API, racontée sur le mode humoristique.
- « Exemples pour BDD » : divers conseils de programmation Basic sur les bases de données et formulaires.
- « Manuel de programmation Basic StarOffice 7 » : ce document écrit par Sun Microsystems vous fournit une autre vision de la programmation de StarOffice, un dérivé d'OpenOffice.org. Vous y trouverez une description succincte, mais couvrant l'ensemble OOoBasic et API, avec quelques informations non décrites ici et des conseils pour un programmeur Visual Basic for Application.

La page Macros du site français d'OpenOffice.org met à disposition un grand nombre d'exemples de macros sélectionnés pour leur valeur explicative.

▸ http://fr.openoffice.org/Documentation/Macros/indexmac.html

Le document « Éléments de programmation des macros dans OOo » est une traduction française d'un document anglais écrit par Andrew Pitonyak. Il contient un grand nombre de « recettes » sur des aspects assez spécialisés de la programmation OpenOffice.org. La traduction étant assez en retard sur le document original, consultez plutôt la version anglaise si vous maîtrisez la langue de Mark Twain.

▸ Français : http://fr.openoffice.org/Documentation/Guides/Indexguide.html
Anglais : http://www.pitonyak.org/oo.php

Le site de langue anglaise OOoMacros contient de nombreuses macros, la plupart assez, voire très élaborées. On y trouve notamment des macros écrites par Danny Brewer, qui sont des modèles de bonne programmation. Vous y retrouverez aussi certaines macros françaises publiées en version anglaise.

- http://ooomacros.org/

Le site de langue anglaise OOoForum consacre la section Code Snippets à des exemples de codage, la plupart en OOoBasic. Ces codages ont été choisis pour leur intérêt pédagogique et technique. Danny Brewer est un des modérateurs de OOoForum.

- http://www.oooforum.org/forum/viewforum.php?f=11

Le projet API du site anglais OpenOffice.org, n'est pas en reste et publie les *Snippets* que lui fournissent les programmeurs API.

- http://codesnippets.services.openoffice.org/

Le projet DBA d'OpenOffice.org consacré aux bases de données (en anglais) offre un fichier ZIP contenant quelques macros d'accès aux bases de données. Décompressez ce ZIP et utilisez Outils > Macros > Macro pour insérer une bibliothèque, qui aura pour nom `DBATools`.

- http://dba.openoffice.org/downloads/DBATools.zip

Votre application OpenOffice.org elle-même est livrée avec plusieurs bibliothèques de macros dans `soffice`. Ces codages sont malheureusement très peu documentés et souvent anciens. On peut y glaner des séquences intéressantes, et même utiliser des macros telles quelles. Voyez notamment les bibliothèques `ImportWizard`, `Gimmicks` et `Tools`.

D'une manière plus générale, les solutions à base de macros citées dans le chapitre 1 du présent ouvrage peuvent aussi apporter des solutions à vos besoins.

Sites Web et forums

Parmi les sources d'information, nous citerons des forums en langue française, anglaise ou autres. Rappelons quelques principes communs à tous les forums :

- Commencez par observer les échanges de messages avant de participer activement ; ceci afin de cerner le domaine des discussions et le niveau technique.
- Ne posez pas une question qui a déjà reçu une réponse récemment ; il existe des moyens de recherche dans chacun des forums.

- Restez poli, évitez le bavardage intempestif, soyez bref et clair dans votre demande.
- Il est très mal élevé de poser dans un forum une question dans une autre langue que celle pour laquelle il est créé.
- Rappelez-vous que ces forums sont animés par des bonnes volontés, qui ne sont pas rétribuées pour cela.

À RETENIR S'inscrire à un forum

Les forums cités sont pour la plupart des listes de distribution de messages (*mailing lists*). Pour savoir comment s'y abonner (gratuitement), envoyer un message, se désabonner, il suffit d'envoyer un courrier électronique vide à l'adresse xxx-info@yyyy, par exemple prog-info@fr.openoffice.org. Un robot vous renverra un message explicatif, en anglais. Comme c'est un robot, inutile d'être poli avec lui, et inutile d'espérer une réponse personnalisée.

En français

Le site français d'OpenOffice.org, http://fr.openoffice.org/ est probablement le plus vivant et le plus complet de tous les sites nationaux d'OpenOffice.org. Nous en avons cité plusieurs pages. Ce site est géré par Sophie Gautier, qui est aussi la modératrice des forums en langue française. Les questions (et réponses) concernant OOoBasic et l'API ont leur place sur la liste prog@fr.openoffice.org. Il arrive que certains messages soient signés de l'un ou l'autre des auteurs de ce livre.

- http://fr.openoffice.org/servlets/ProjectMailingListList

Le site OOoConv, créé par un des auteurs, présente plusieurs outils dont l'analyse sera instructive : OOoConv, FitOO, BatchConv.

- http://oooconv.free.fr/

En anglais

Le site http://www.oooforum.org/ a une vitalité semblable aux forums français d'OpenOffice.org, mais il n'est accessible qu'avec un navigateur Web. On y trouve en particulier :

- le forum Macros and API, supervisé par Danny Brewer, où vous trouverez de nombreuses informations dans les réponses déjà faites ;
- le pseudo-forum Code Snippets, déjà cité, qui n'est pas un vrai forum de discussion mais un répertoire d'exemples de codage.

La page http://www.openoffice.org/mail_list.html vous donne accès à tous les forums OpenOffice.org de langue anglaise. Le forum dev@api.openoffice.org est consacré aux questions sur l'API elle-même, posées par des programmeurs expérimentés. Les réponses sont souvent

apportées par les chefs développeurs d'OpenOffice.org dans la mesure de leur disponibilité. Abstenez-vous d'y poser des questions de débutant.

Pour les programmeurs aguerris, le site du projet UDK d'OpenOffice.org http://udk.openoffice.org/ est consacré aux développements permettant de programmer OpenOffice.org dans différents langages, dont OOoBasic. Il fournit différents liens et des pages explicatives. Ce projet possède un forum dev@udk.openoffice.org. Il traite de l'interfaçage de langages de programmation (OOoBasic, Python, Java, etc) avec l'API OpenOffice.org, et de OOoBasic.

Autres langues

Le site principal d'OpenOffice.org possède de plus en plus de subdivisions nationales, accessibles depuis la page principale (voir la liste déroulante Native Language Projects). Il faut reconnaître qu'elles n'offrent que peu d'informations nouvelles sur la programmation d'OpenOffice.org.

- http://www.openoffice.org/

Le site privé allemand Kienlein offre diverses informations de programmation d'OpenOffice.org pour accéder à des bases de données.

- http://kienlein.com/pages/oo.html

IssueZilla

IssueZilla, appelé aussi en abrégé IZ, est une base de données pour gérer les rapports d'anomalies et de demandes d'amélioration. Également connue depuis peu sous le nom de `IssueTracker`, IT en abrégé, elle est consultable sur Internet. Les rapports IssueZilla sont rédigés en anglais, afin d'être compréhensibles par des lecteurs du monde entier. Il est donc nécessaire d'avoir une connaissance de cette langue pour utiliser cette source d'information.

Rechercher un rapport dans IssueZilla

La page d'entrée pour rechercher quoi que ce soit dans IssueZilla est :

- http://www.openoffice.org/project/www/issues/query.cgi

Chaque rapport IssueZilla reçoit un numéro. Pour afficher un rapport dont on connaît le numéro, il suffit de remplir le champ en haut de la page et de cliquer sur le bouton Jump to Issue.

Si vous cherchez s'il existe un rapport sur un sujet donné, il faut remplir certains champs proposés par le formulaire. Plus vous remplissez de champs, plus la recherche se focalise. La difficulté est de se demander quels mots ont pu être utilisés dans les rapports déjà écrits. Pour éviter des recherches infructueuses, il est préférable de commencer par une recherche assez générale, un mot ou deux dans le titre du rapport, et de restreindre la recherche si elle renvoie un grand nombre de rapports. Souvent il faut essayer des mots synonymes, ou écrits différemment.

Rédiger un rapport

De nombreux rapports sont écrits chaque jour et les développeurs ont du mal à les analyser. Vous ne devriez en écrire que sur un sujet que vous estimez important et faire une recherche préalable pour éviter de dupliquer un rapport existant.

> On peut écrire un rapport pour seulement deux raisons : demander une amélioration ou signaler une anomalie. IssueZilla n'est pas un site pour demander une aide, voyez les forums pour cela.

Les demandes d'améliorations sont lues par les développeurs et classés avec un horizon de prise en compte éventuelle (target milestone). Ceci ne veut pas dire qu'elles seront introduites, car d'autres critères entrent en jeu (ressources, intérêt marketing, complexité).

Avant d'écrire un rapport d'anomalie, vérifiez très soigneusement qu'il s'agit bien d'une erreur de l'application OpenOffice.org et non pas d'une erreur de compréhension de votre part, ou une mauvaise configuration de votre ordinateur. Simplifiez au maximum les conditions d'apparition de l'anomalie. Joignez si possible un document démontrant l'erreur, par exemple une macro réduite au codage minimum nécessaire. En effectuant ce travail, vous analyserez mieux l'anomalie et souvent vous verrez qu'elle provient d'une erreur de votre part. Soyez le plus clair possible dans les explications, restez factuel. Pensez que les développeurs ont à lire des dizaines de rapports, en plus de leur activité habituelle.

L'écriture de rapports est réservée aux membres d'OpenOffice.org. Ce n'est pas une société secrète, n'importe qui peut devenir membre, gratuitement et sans engagement. Allez à la page http://www.openoffice.org/servlets/Join et remplissez le formulaire. Vous avez maintenant une identité dans le système et un mot de passe.

Pour créer un rapport, vous devez d'abord vous connecter au système, en allant à la page http://www.openoffice.org/servlets/TLogin. Puis, commencez à la page http://www.openoffice.org/issues/enter_bug.cgi. Une fois choisi le domaine principal, vous vous retrouvez devant un formulaire dont il faut remplir au mieux les cases. Ce n'est pas facile les premières fois. Le titre du rapport et le texte explicatif doivent obligatoirement être en anglais. Envoyez le rapport. Un numéro sera attribué automatiquement et vous recevrez un courrier électronique pour chaque évolution de ce rapport. Utilisez la page Web renvoyée pour ajouter éven-

tuellement un fichier. Plus tard, en réaffichant le rapport, vous pourrez ajouter de nouvelles informations, à condition de vous être identifié. Soyez très patient, un rapport peut rester plusieurs mois sans réponse...

Si votre anomalie est reconnue comme telle, la date prévisionnelle de correction sera indiquée dans le champ Target Milestone. Les ressources humaines étant toujours limitées, il est encore possible que cette date soit repoussée à plus tard lors d'une ré-analyse des priorités.

Partager la connaissance

Comme nous l'avons vu dans l'exposé précédent, la communauté est très active en ce qui concerne l'API et les macros. De nombreuses sources sont disponibles et il ne tient qu'à nous, acteurs et utilisateurs, d'enrichir notre connaissance commune.

Vous venez d'écrire une macro qui comble un manque d'OpenOffice.org ou simplement se révèle utile au jour le jour ; il est fort probable que cette macro puisse servir à quelqu'un d'autre. Comme nous apprenons tous de la lecture des macros publiées par d'autres, vous pouvez vous aussi participer en rendant public votre travail.

Vous serez accueillis avec enthousiasme sur les listes de diffusion francophones comme doc@fr.openoffice.org ou prog@fr.openoffice.org. Votre contribution y sera valorisée et ce, en échangeant juste quelques courriers électroniques. Pourquoi s'en priver ?

Conclusion

La richesse des ressources disponibles sur l'Internet, source d'informations dynamique et vivante, est illimitée et nous n'en présentons qu'une partie. N'hésitez pas à en tirer régulièrement profit.

Index

Symboles

" 75
158
& 75, 144
' 61
+ 76, 144
: 62
= 63
_ 61

A

Abs 147
Access 614
actions utilisateur 651
Add-on 705
adresses
  de fichiers 153
  URL 181
adresseString 694
adrZoneString 694
affichage 312, 407, 650
alphaXY 694
And 82
and 84
Andrew Pitonyak 711
API 673
application associée 648
arguments 63
Array 99
arrière-plan 399
  de page 402
arrondir 146
As 87
As New 183
Asc 145
assistant de création d'états 561
Atn 141
AutoText 642
avancement (état) 652

B

barre de défilement 528
Base 102
base de registres 653
BasicLibraries 134, 666
BeanShell 54
Beep 162
bibliothèque 132, 500
  ajouter 42
  copier 43
  protéger 43
  renommer 41
bibliothèque Standard 34, 41
Bibliothèques 35, 39
Blue 153
boîte de dialogue 35, 39
  créer/supprimer 44
boîte de dialogue *Voir* dialogue
Boolean 81
bordure 253, 265, 281, 291, 292, 297, 331
bouton par défaut 513
Byte 674
ByVal 124

C

Calc
  CALCSQL1 699
  CALCSQL2 699
  colonne 338
  configuration 374
  currentSelection 326
  diagrammes 377
  DrawPage 474, 482
  en-tête 387
  feuille 315
  feuille visible 319
  figer ligne/colonne 327
  fonction 370, 372
  fonctions de zone 354
  format de nombre 347
  formes 474
  getCellByPosition 321
  getCellRangeByName 321, 322
  getCellRangeByPosition 322
  hypertexte 374
  images 482
  imprimer 375
  ligne 337
  mot de passe 320
  options de Calc 389
  page de dessin 474, 482
  pied de page 387
  protéger feuille 320
  recalculer 352
  rechercher 361
  recopier formule 360
  remplacer 361
  sélection 325
  tableau 355
  trier 366
Call 120
caractère
  accentuation 229
  aucun formatage 232
  barré 231
  casse 230
  clignoter 231

contour 231
couleur 230
exposant 230
gras 227
indice 230
italique 228
localisation 231
non imprimable 145
ombre 231
police 231
relief 229
souligné 228
style 226, 231
taille 231
Case 106
casse 62, 147
CBool 149
CCur 146
CDate 148
CDateFromIso 148
CDateToIso 148
CDbl 146
cellule (Calc)
bordures 331
cellule 321
contenu 339
coordonnées 323, 693
copier 358
curseur d'écriture 340
déplacer 357
effacer zone 352
format de caractère 346
insérer
paragraphe 344
texte 343
note 335
propriétés 328
protection 330
style 345
zone 322
nommée 324
CenterDialog 699
certificats numériques 16
chaîne de caractères (fonctions) 138
chaînes longues 700
champ
date 514, 624, 629
de table 576
de texte 298
formaté 520, 624, 629
horaire 516, 624, 629
masqué 519, 624, 629
monétaire 517, 624, 629
numérique 503, 624, 629
utilisateur 207
changement de casse 147
CharacterCount 208
charger une bibliothèque 35, 133
ChDir 154
ChDrive 154
chiffrer une macro 36, 39
Choose 109
CInt 146
CLng 146
Close 158
Code Snippets 712
collection 676
CollerDepuisPressePapier 702
coloration syntaxique 47
COM 685
commentaires 61
CompatibilityMode 73
compiler 51
Config 642
Const 91
constante nommée 674
conteneur de bibliothèques 35, 36
conteneur soffice 40
contrôle
Voir dialogue, formulaire
conversions
automatiques 143
explicites 145
ConvertFromURL 153
ConvertToURL 153
coordonnées 323, 693
copier un tableau 97
CopierDansPressePapier 702
Cos 141
couche 403, 407, 414
couleurs 152
courrier électronique 656
CR 145
createEnumeration 225, 353, 564
CreateUnoDialog 499, 678
CreateUnoListener 678
CreateUnoService 678
CreateUnoStruct 678
CreateUnoValue 678
CSng 146
CStr 146
CSV 197
CurDir 154
Currency 79
CurrentController 233, 244
CurrentSelection 411
curseur 215, 232, 236, 308
d'écriture 215, 266, 282, 340, 432
de cellule 261

D

DannyB 710
Date 86
Date (fonction) 142
DateSerial 86, 148
DateValue 148
Day 148
Defxxx 101
Delphi 686
démarrage 135
Developer's Guide 687
DialogLibraries 499
Dialogue 699
dialogue 493
à pages multiples 547
barre de défilement 528

barre de progression 529
bouton 497
bouton radio 509
Button 497
case à cocher 508
champ
  date 514
  formaté 520
  horaire 516
  masqué 519
  monétaire 517
  numérique 503
CheckBox 508
choix 1 parmi N 509
Compteur 503
contrôles 495
CurrencyField 517
DateField 514
emboîté 548
Etiquette 496
FileControl 526
FormattedField 520
image 527
ImageControl 527
Label 496
ligne
  horizontale 514
  verticale 514
ListBox 504
modèle 533
modification
    dynamique 534
NumericField 503
OptionButton 509
PatternField 519
ProgressBar 529
ScrollBar 528
sélection de fichiers 526
TextField 501
TimeField 516
traiter (événements) 540, 541
zone
  de groupe 513
  de liste 504
  de texte 501
Dim 71
DimArray 101
Dir 154
division
  arrondie 77
  entière 77
Do 114
document
  en cours 180
  modèle 186
documentation 687
documenter une macro 707
Double 77
Draw
  arrière-plan 399
    de page 402
  configuration 462
  couche 403
    propriétés 407
  getByIdentifier 448
  image 455, 477
  imprimer 461
  mode d'affichage 407
  page 394
    propriétés 408
    visible 398

E

écrire un fichier 157
écriture dynamique 663
EDI 33, 46
éditeur de macros 46
élévation à la puissance 78
ElseIf 105
Empty 88
End Sub 119
enregistrer sous 555
enregistreur de macros 17
en-tête 296
Environ 162
Environnement de Développement Intégré 33
Eof 159, 162
EqualUnoObjects 149, 242, 679
équations 486
Eqv 83
ergonomie 511
Erl 169
Err 169
erreur (informations) 169
erreurs (messages) 679
erreurs d'exécution (liste) 169
Error 169
étiquette 115
événements 540, 632
  informations 543
exécuter depuis l'éditeur de macros 48
exécution étape par étape 50
Exit 114
Exit Function 132
Exit For 112
Exit Function 168
Exit Sub 127, 168
Exp 142
Explicit 70
exponentielle 142
exporter en CSV 197

F

False 81
fermer le document 189
feuille 315, 319, 320
fichier 154
  binaire 644
  temporaire 642
  texte 158
  ZIP 646, 699
FileAttr 162
FileCopy 154
FileDateTime 154
FileExist 154
FileLen 154
FilePicker 552
filtre CSV 198

filtres 193
FindChartByObjName 696
FindCtrlShapeByName 699
FindObjectByName 695
Fix 146
Flash 195
Focus 512
FolderPicker 553
fonction 370, 372
fonctions
  trigonométriques 141
For 111
format 346, 347
Format (fonction) 141
forme (Draw)
  Bézier 445
  carré 434
  cercle 434
  cisaillement 428
  combiner 454
  connecter 454
  connecteur 440
  contour 416
  couche 414
  couleur 416, 419
  curseur d'écriture 432
  dégradé 419
  dessiner 409
  ellipse 434
  étiquette 442
  exporter 455
  fond 418
  grouper 453
  hachuré 424
  ligne
    brisée 437
    de cote 443
    simple 436
  motif bitmap 426
  ombre 427
  ordre Z 453
  points de colle 448
  polygone 438
  position 414
  du texte 430
  rectangle 434
  rotation 428
  sélection utilisateur 411
  sélectionner 412
  supprimer 413
  taille 414
  texte 428, 435
    animé 431
  transparence 422
  trouver 410
  type 414, 434
formulaire 625
  accéder 618
  bouton 622
  case à cocher 623
  champ
    date 624, 629
    formaté 624, 629
    horaire 624, 629
    masqué 624, 629
    monétaire 624, 629
    numérique 624, 629
  choix 1 parmi N 623
  FindCtrlShapeByName 620
  Grid 625
  grille 625
  image 625
  sélection de fichier 625
  table 625
  zone
    de liste 622, 631
    de texte 624, 629
formule 486
forum 712
FreeFile 158
Function 130
fusionner cellules 271

G

Gallery 642
geler l'affichage 650
gestion de fichiers 154
Gestion de macros 41
gestionnaire de sources de
  données 560
Get 158
GetAttr 157
getDirectory 702
getFileExt 702
getFileNameOnly 702
getFullFileName 702
getIndexByName 693
getLocaleStyleName 698
getParentDir 702
getPathSeparator 154
GetProcessServiceManager 678
getPropertyValue 676
getPropVal 203, 692
GetSolarVersion 162
GetSystemTicks 142
Global 74, 92, 94
GlobalScope 135, 500
Gosub 128
GoTo 115
Green 153

H

hasard 142
hasLocation 188
HasUnoInterfaces 679
-headless 28
Help 642
Hex 146
Hidden 184
Hour 148
HTML 195
hyper 674

I

IDL 687
If Then Else 103
IIf 108
image 476
  propriétés 484
Imp 83
import/export 193
importer du CSV 197

Impress
configuration 466
exécuter macro 464
impression 466
page
de notes 464
prospectus 465
styles 465
transitions 467
imprimer 375, 461
un document 200
indentation automatique 48
informations du document 205
Input 158
InputBox 152
insertString 220
installation 641
InStr 138
Int 146
Integer 76
InteractionHandler 185
interception d'erreur 165
interface 674
IsArray 89, 149
IsDate 89, 149
IsEmpty 89, 149
IsMissing 122, 149
isModified 187
IsNull 89, 149
IsNumeric 89, 149
IsObject 89, 149
IssueZilla 714
IsUnoStruct 149, 679

J
Java 56, 675
JavaScript 53
Join 140

K
Kill 154

L
lancer un programme 163
langue 232, 347
LBound 96
Lcase 147
Left 139
Len 138
LF 145
licence IX, 691
ligne de commande (exécuter une macro) 28
lignes (de codage) 61
Line Input 159
lire un fichier 157
Listener 632, 659
LoadComponentFromURL 181
LoadLibrary 134
Loc 162
Locale 208
lockControllers 651
Lof 162
Log 142
logarithme népérien 142
Long 76
Loop 114
LSet 141
LTrim 139

M
macro
bibliothèque 17, 34
bouton de barre d'outils 19, 24
dossier Mes Macros 25, 28
dossier share 23
dossier user 23
entrée de menu 20, 25
événement 21, 26
exécuter en ligne de commande 28
Macros OpenOffice.org 21, 23, 25, 28
menu Outils 18, 21
module 17, 34
raccourci clavier 18, 23
Map AppFont 494
méthode 179, 674
Mid 139
Minute 148
mise au point 49
MkDir 154
mod 78
module 40
créer/supprimer 44
déplacer 44
renommer 51
Month 148
mot de passe 183
MsgBox 149

N
Name (fonction) 154
Next 111
nombre
aléatoire 142
hexadécimal 146
octal 146
noms de variables 62
Not 82
not 84
nouveau document 186
Now 142
Null 89

O
Object 87, 179
Oct 146
OLE 149, 311, 695
On Error GoTo 166
On Error Resume Next 168
On Goto 115
OOoBasic 8
OOoMacros 712
Open 157
opérateurs
booléens 82
de comparaison 84
mathématiques 77, 78
Optional 122
Or 83, 84

ordre d'évaluation 81, 85

P

palette des couleurs 649
par valeur 124
ParagraphCount 208
paragraphe 221, 223, 225, 248, 290
paramètres
  de sous-programme 121
  optionnels 122
parenthèses 63
PATH 643
pays 232, 347, 643
PDF 195
Pi 141
pile des appels 52
point d'arrêt 51
portée des variables 72, 125
Preserve 95
Print 151, 158
printProps 203, 692
Private 73, 92, 94, 120
PropertyValue 183
propriété 179
protection 330
pseudo-propriété 676
Public 74, 92, 94, 132
Put 158
Python 55, 684

R

racine carrée 142
radians 141
Random Access 161
Randomize 142
rechercher 241, 361
  une chaîne 138
récursion 155
Red 153
Redim 94
RemplaceChaine 701
remplacer 245, 361
répertoire 154
répertoire de confiance 12
requête 580, 581, 584, 596, 602, 611, 614
Reset 162
resizeImageByDPI 697
resizeImageByWidth 697
reste de la division entière 78
Resume etiquette 167
Resume Next 166
Return 128
RGB 152
Rhino 53
Right 139
RmDir 154
Rnd 142
RSet 141
RTrim 139

S

saut
  de colonne 222
  de ligne 221
  de page 222
sauver un document 188
script non Basic en ligne de commande 32
SDK 687
Second 148
sécurité (des macros) 11
Seek 162
Select Case 106
sélection 234, 264, 325, 411
  de fichier 552
  de répertoire 553
séquence 674
serial number 148
service 674
services de dialogues 552
SetAttr 157
setPropertyValue 676
setPropVal 204, 692
Sgn 141
Shell 163
signature numérique 16
signatures numériques 16
signet 306
Sin 141
Single 77
site Web 712
soffice 133, 712
soffice.exe 28
sources de données
  champs de table 576
  créer 566
  créer une requête 581
  créer une table 578
  enregistrement
    effacer un 594
    insérer un 589
    modifier un 593
  événements 602
  filtre 601
  getXXX 587
  listener 602
  lister 563
  mot de passe 572
  propriétés 565
    de connexion 573
  publipostage 609
  requête
    dans Calc 611
    MS-Access 614
    paramétrée 596
    pré-enregistrée 580
  résultats de requête 584
  ResultSet 586
  RowSet 597
  se connecter 570
  setXXX 597
  supprimer 566
  tables 574
  transactions 605
  tri 600
  updateXXX 593
  utiliser requêtes pré-enregistrées 602
  valeurs nulles 596
  VBA 614

sous-formulaires 625
Space 140
Split 140
Sqr 142
StarBasic 8
StarDesktop 181, 677
Static 126
Step 111, 547
storeAsURL 188
Str 146
StrComp 138
String 75
String (fonction) 140
structure 179
style 287, 380, 459
  cadre 296
  caractère 226, 292
  cellule 383
  copier 289
  créer 383, 461
  DisplayName 288
  modifier 460
  numérotation 296
  page 292, 383
  paragraphe 225, 290
Sub 119
supportsService 225, 326, 679
Switch 110
syntaxe 51

T

tableau 92
  Calc 355
  de propriétés 691
  de Variant 98
  irrégulier 99
  Writer 249
tabulation 145
Tan 141
témoin 49
Temp 642
Template 186, 642
TextEmbeddedObjects 469
TextFrames 469
TextGraphicObjects 469
ThisComponent 181, 677
Time 142
Time_HMS 700
Timer 142
TimeSerial 86, 148
TimeValue 148
tiret 221
to 93
transaction 605
trier 366
  un tableau 267
Trim 139
TriShell 701
True 81
TwipsPerPixelX 163
TwipsPerPixelY 163
TypeName 90

U

UBound 96
Ucase 147
Unicode 76, 84, 145
unlockControllers 651
Until 114

V

Val 145
valeur
  de propriété 183
  initiale (des variables) 72
variables 69
  statiques 126
Variant 87
Variant et tableaux 98
VarType 90
VBA (Visual Basic for Application) 9, 711

W

Wait 142
WeekDay 148
Wend 113
While 113
With 207, 239, 241, 280, 585
WordCount 208
WordSeparator 208
Work 642
Write 158
Writer
  appliquer un style 225
  bookmark 306
  bordure 253, 265, 281, 291, 292, 297
  cadre 273
  cellule 265
  champ de texte 298
  CharStyleName 226
  colonne de tableau 259
  compareRegionEnds 237, 244
  compareRegionStarts 237
  configuration d'affichage 312
  createTextCursor 215
  createTextCursorByRange 217, 234
  CurrentController 237
  CurrentSelection 235
  curseur
    d'écriture 215, 266, 282
    de cellule 261
    déplacer 215
    où est-il ? 236
    visible 232, 308
  DrawPage 470, 479
  en-tête 296
  espace insécable 221
  formater 225
  formes 470
  formule dans une cellule 267
  fusionner cellules 271
  gotoRange 218
  hypertexte 309
  images 479
  insérer texte 219
  insertControlCharacter 221
  insertDocumentFromURL 223

isCollapsed 234
ligne de tableau 258
note 302
page de dessin 470, 479
paragraphe 221, 223, 248
ParaStyleName 225
pied de page 296
rechercher 241
remplacer 245
retour à la ligne 221
saut
  de colonne 222
  de ligne 221
  de page 222
scinder cellule 272
section 283
sélection 234, 264
signet 306
tableau 249
tabulation 221, 239
taquet de tabulation 239
Text 214
texte 234
texte conditionnel 304
TextField 298
tiret
  conditionnel 221
  insécable 221
trier un tableau 267
WordSeparator 217
zone de texte 216, 218, 233

X

Xor 83
xor 84
XRay 688
Xray 707

Y

Year 148

Z

ZIP 646
zone 322, 352
  de groupe 513
  de liste 504, 622, 631
  de texte 216, 218, 233, 501, 624, 629
  fonctions 354
zoom 209

Programmation OpenOffice.org 2

OpenOffice.org, suite bureautique libre et gratuite, est munie du langage de script OOoBASIC et d'une API permettant de manipuler ses objets. Ainsi automatisable et extensible, elle peut s'intégrer parfaitement au système d'information de l'entreprise.

Le livre de référence sur la programmation OpenOffice.org et StarOffice

Écrit par deux contributeurs majeurs de la communauté francophone fr.OpenOffice.org, ce livre est une référence incontournable sur le puissant langage de macros OOoBASIC et sur l'API d'OpenOffice.org. Destiné aussi bien aux utilisateurs d'OpenOffice.org que de StarOffice, il explique comment gérer des fichiers OOo, automatiser des tâches répétitives, traiter des chaînes, créer des boîtes de dialogue et des formulaires, accéder et exploiter des bases de données, intercepter des événements, explorer et utiliser l'API d'OpenOffice.org, créer dynamiquement des macros, gérer des erreurs…

Nouveautés liées à la version 2.0 d'OpenOffice.org

La version 2 d'OpenOffice.org comporte certaines nouveautés du point de vue de la programmation. Elles sont soulignées tout au long du livre et, au besoin, accompagnées d'exemples dédiés.
Cet ouvrage aborde notamment l'interface utilisateur, la sécurité des macros, les autres langages de script (JavaScript, BeanShell, Python). Il décrit également l'environnement de développement, l'installation d'add-ons, les appels de scripts, et explique les particularités de la version 2 d'OpenOffice.org pour les sources de données, requêtes, rapports et formulaires de Base.

Au sommaire

OOoBasic • Domaine d'utilisation et comparaison avec VBA • Écrire, enregistrer, exécuter des macros • Sécurité • Exécution en ligne de commande • Bonnes pratiques de programmation Basic • Environnement de développement en JavaScript, BeanShell ou Python • Types, variables et tableaux • Conditions, boucles et branchement • Sous-programmes Sub et Function • Bibliothèques • Principales instructions de traitement • Chaînes de caractères • Fonctions numériques • Date et heure • Fonctions de conversion • Interface utilisateur : écran, clavier • Traitement des fichiers • Fonctions système • Traitement des erreurs • **Les documents OpenOffice.org** • Accéder au document • Enregistrer • Filtres d'import/export • Imprimer • Configurer un document • **Documents Writer** • L'objet Text • Le curseur • Insérer et supprimer • Appliquer une mise en forme • Curseur visible et sélection • Rechercher, remplacer • Tableaux, cadres, sections et styles • En-têtes et pieds de page • Champs de texte, champs utilisateur et textes conditionnels • Signets et renvois • Configurer l'affichage • **Documents Calc** • Feuille, cellule, zones, lignes et colonnes • Lire et écrire dans une cellule • Rechercher, remplacer, trier • Créer une fonction. • Formats de nombre • Zones visibles • Énumérer les cellules d'une zone • **Document Draw et Impress** • Pages de dessin, arrière-plans et couches • Les formes • Points de colle • Export • Insertion d'objets • **Les boîtes de dialogues** • Construire une boîte de dialogue avec l'EDI • Champs de saisie • Aspect visuel et notion de Focus • Modifier dynamiquement un contrôle • Traiter des événements • Pages multiples et dialogues emboîtés • **Les sources de données** • Outils intégrés • Accès aux données avec SQL • RowSet • Transactions • **Les formulaires** • Contrôles et base de données • Principes communs aux contrôles • Contrôle Table • Formulaires intégrés dans un document Base • **Techniques avancées** • Répertoires d'installation • Gestion des fichiers • Accéder à la base de registres de MS-Windows • Envoi d'un document par e-mail • Écrire dynamiquement des macros • **Comprendre l'API** • Messages d'erreur OOoBasic • L'API avec JavaScript, BeanShell, Python, COM • **Routines utilitaires** • **Ressources Internet** • Outils pour le développement • IssueZilla.

À qui s'adresse cet ouvrage ?

– Aux utilisateurs d'OpenOffice.org et StarOffice souhaitant automatiser et étendre leur suite bureautique ;
– À ceux qui migrent vers OpenOffice.org et souhaitent recréer des macros existantes ;
– Aux développeurs d'applications d'entreprise et services informatiques ayant à intégrer la suite OpenOffice.org ;
– Aux étudiants et tous ceux souhaitant s'initier à l'API d'OpenOffice.org et à son langage de macros OOoBASIC.



Bernard Marcelly
Laurent Godard

Bernard Marcelly a effectué sa carrière dans la R&D de grands projets logiciels de télécommunications. Parmi ses nombreuses contributions au projet OpenOffice.org, un document d'introduction à l'API, des macros et des outils pour faciliter la programmation d'OOo.

Laurent Godard est l'auteur d'outils Open Source incontournables tels DicOOo, FontOOo, OOoConv, fruits de son expertise de l'API d'OpenOffice.org. Il est représentant élu des contributeurs de la communauté OpenOffice.org et responsable du projet Extensions. Il est enfin directeur technique d'Indesko, société offrant des solutions libres pour le poste de travail et l'informatique d'entreprise.

Les auteurs, tous deux membres actifs de la communauté OpenOffice.org, ont choisi de reverser leurs droits à CUsoon (cusoo.org), association loi 1901 qui réunit les clubs d'utilisateurs de StarOffice et d'OpenOffice.org.

Recommandé par fr.openoffice.org

OpenOffice.org

Conception : Nord Compo