B

Routines utilitaires

La plupart des sous-programmes présentés dans ce chapitre ont été utilisés dans les exemples de ce livre. Ils pourront vous être très utiles dans vos projets de macros.

Tableaux de propriétés

Ce que nous appelons tableau de propriétés est un tableau unidimensionnel dont chaque élément est une structure UNO `com.sun.star.PropertyValue`. Un tel tableau est souvent utilisé pour transmettre des informations à l'API OpenOffice.org.

Créer un tableau de propriétés

Le chapitre 7 utilise la fonction `CreateProperties` dans plusieurs exemples, notamment pour l'export PDF. Cette fonction sert à créer en une instruction un tableau de propriétés qui sera utilisé comme argument d'une méthode API, ou parfois comme valeur d'une propriété appartenant à un autre tableau de propriétés. Elle comporte un seul argument qui est un tableau Basic à une dimension, créé dynamiquement par la fonction Basic `Array`. Les arguments de la fonction `Array` sont, successivement, le nom et la valeur de chaque propriété du tableau à construire.

```
rem CodeAnnexeB-01.odt   bibli : Proprietes Module1
Option Explicit
```

```
Function CreateProperties(propList() As Variant) As Object
Dim n as long, x as long

n = UBound(propList)
if n < 0 then
  CreateProperties = Array()
else
  if (n and 1) = 0 then
    MsgBox("Erreur : nombre impair d'arguments", 16, "CreateProperties")
  else
    Dim p(n\2) As New com.sun.star.beans.PropertyValue
    for x = 0 to n\2
      p(x).Name = propList(2*x)
      p(x).Value = propList(2*x +1)
    next
    CreateProperties = p()
  end if
end if
End Function
```

Accéder à une propriété par son nom

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.

Ces routines adressent une propriété dont le nom est donné en argument. Rappelons que la casse doit être respectée pour ce nom. La fonction `hasProp` renvoie `True` si la propriété existe dans le tableau. La fonction `getPropVal` renvoie la valeur de la propriété. La routine `setPropVal` affecte une valeur à la propriété. Ces deux dernières routines déclenchent une erreur s'il n'existe pas dans le tableau de propriété du nom indiqué.

```
rem CodeAnnexeB-01.odt   bibli : Proprietes Module2
Option Explicit

' renvoie True si une propriété existe au nom indiqué
Function hasProp(descr As Variant, nomProp As String) As Boolean
Dim p As Object
for each p in descr
  if p.Name = nomProp then
    hasProp = True
    Exit Function
  end if
next
hasProp = False
End Function
```

```
' renvoie la valeur de la propriété nomProp
Function getPropVal(descr As Variant, nomProp As String) As Variant
Dim p As Object
for each p in descr
  if p.Name = nomProp then
    getPropVal = p.Value
    Exit Function
  end if
next
' la propriété nomProp n'existe pas !
err = 423 ' déclencher erreur : Propriété ou méthode introuvable
End Function


' affecte la valeur valProp à la propriété nomProp
Sub setPropVal(descr As Variant, nomProp As String, valProp As Variant)
Dim p As Object
for each p in descr
  if p.Name = nomProp then
    p.Value = valProp
    Exit Sub
  end if
next
' la propriété nomProp n'existe pas !
err = 423 ' déclencher erreur : Propriété ou méthode introuvable
End Sub
```

Coordonnées de cellules

L'API utilise des coordonnées de cellules en X et Y, débutant à zéro. L'utilisateur emploie plutôt des coordonnées alphanumériques comme A1.

- La fonction `adrZoneString` renvoie la chaîne de caractères correspondant aux coordonnées de l'objet `RangeAddress` passé en argument, y compris le nom de la feuille.
- La fonction `adresseString` effectue le même travail pour un objet `CellAddress`.
- La fonction `nomColonne` renvoie le nom de la colonne (A, B, ... CF...) correspondant à l'index de colonne en argument.
- La fonction `indexColonne` renvoie l'index de la colonne à partir de son nom (A, B, ... CF...) ; cela permet de clarifier les codages en utilisant explicitement les noms de colonne.

Ces fonctions déclenchent une erreur si l'argument d'adresse est incorrect.

```
rem CodeAnnexeB-01.odt   bibli : Cellules Module1
Option Explicit


' renvoie l'adresse textuelle d'une RangeAddress
Function adrZoneString(monCalc As Object, adrZone As Object) As String
Dim z As Object
On Error GoTo horsLimite
z = monCalc.Sheets(adrZone.Sheet).getCellRangeByPosition(_
  adrZone.StartColumn, adrZone.StartRow, _
  adrZone.EndColumn, adrZone.EndRow)

adrZoneString = join(split(z.AbsoluteName, "$"), "")
On Error GoTo 0
Exit Function
horsLimite:
  Resume horsLimite2
horsLimite2:
  On Error GoTo 0
  err = 14 ' adrZone contient une valeur hors limites
End Function


' renvoie l'adresse textuelle d'une CellAddress
Function adresseString(monCalc As Object, adrCell As Object) As String
Dim c As Object
On Error GoTo horsLimite
c = monCalc.Sheets(adrCell.Sheet).getCellByPosition( _
  adrCell.Column, adrCell.Row)

adresseString = join(split(c.AbsoluteName, "$"), "")
On Error GoTo 0
Exit Function
horsLimite:
  Resume horsLimite2
horsLimite2:
  On Error GoTo 0
  err = 14 ' adrCell contient une valeur hors limites
End Function


' renvoie le nom d'une colonne à partir de son numéro
Function nomColonne(monCalc As Object, X As Long) As String
Dim uneCellule As Object
On Error GoTo horsLimite
uneCellule = monCalc.Sheets(0).getCellByPosition(X,0)
nomColonne = uneCellule.Columns.ElementNames(0)
```

```
On Error GoTo 0
Exit Function
horsLimite:
  Resume horsLimite2
horsLimite2:
  On Error GoTo 0
  err = 14 ' X contient une valeur hors limites
End Function


' renvoie le numéro d'une colonne à partir de son nom
Function indexColonne(monCalc As Object, col As String) As Long
Dim uneCellule As Object
On Error GoTo horsLimite
uneCellule = monCalc.Sheets(0).getCellRangeByName(col & "1")
indexColonne = uneCellule.CellAddress.Column
On Error GoTo 0
Exit Function
horsLimite:
  Resume horsLimite2
horsLimite2:
  On Error GoTo 0
  err = 14 ' col contient un nom inacceptable
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-01.odt   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`
Contrôle de formulaire	`com.sun.star.drawing.ControlShape`

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`.

Redimensionner une image

Le sous-programme `resizeImageByWidth` redimensionne une image à une largeur donnée, en gardant ses proportions. Après insertion d'une image dans un document,

l'objet image fournit à travers sa propriété `Graphic` un objet donnant des informations sur l'image elle-même. Dans ce dernier objet, la structure `SizePixel` nous donne la taille de l'image en pixels. Une règle de trois nous permet d'en déduire la hauteur nécessaire pour la largeur souhaitée. Pour modifier les dimensions, nous utiliserons la propriété `Size` de l'objet image, qui possède une structure équivalente dans laquelle les dimensions sont mesurées en 1/100 de mm.

```
rem CodeAnnexeB-01.odt   bibli : Images Module1
Option Explicit

Sub resizeImageByWidth(uneImage As Object, largeur As Long)
Dim imageInfo As Object, Proportion As Double, Taille1 As Object
imageInfo = uneImage.Graphic
Taille1 = imageInfo.SizePixel
Proportion = Taille1.Height / Taille1.Width
Taille1.Width = largeur ' largeur en 1/100 de mm
Taille1.Height = Taille1.Width * Proportion
uneImage.Size = Taille1
End Sub
```

Attention

L'image doit d'abord être insérée dans le document pour que `Graphic` fournisse des informations sur l'image.

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. Celle-ci est exprimée en points par pouce (en anglais *DPI*). Connaissant la taille d'un pouce, on peut déterminer les dimensions nécessaires en 1/100 de mm.

```
rem CodeAnnexeB-01.odt   bibli : Images Module1
Option Explicit

Sub resizeImageByDPI(uneImage As Object, DPI As Long)
Dim imageInfo As Object, Proportion As Double, Taille1 As Object
Const pouce = 2540 ' longueur en 1/100 de mm
imageInfo = uneImage.Graphic
Taille1 = imageInfo.SizePixel
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 10).

```
rem CodeAnnexeB-01.odt   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`.

Adresses URL de fichiers et répertoires

Le document CodeAnnexeB-01.odt contient dans la bibliothèque URLaddresses plusieurs fonctions facilitant l'analyse d'adresses de chemins de fichiers. Elles sont écrites en anglais pour des raisons d'universalité. Le codage est un bon exemple de la puissance des fonctions Basic join et split, signalées au chapitre 5.

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, par exemple : monfichier.odt
getFileNameOnly	Chemin d'un fichier	Nom du fichier sans l'extension, par exemple : monfichier
getFileExt	Chemin d'un fichier	Extension du fichier, avec le point, par exemple : .odt

Trier un tableau de données en Basic

Les langages de script Python, Beanshell, JavaScript, disposent d'une fonction intégrée de tri, mais pas OOoBasic. La routine utilitaire QuickSort (tri rapide) effectue le tri d'un tableau (Array) de Variant. Voici la routine principale, avec ses arguments d'appel.

```
rem CodeAnnexeB-01.odt   bibli : Trier Module2
Option Explicit

' myList() : tableau d'éléments à trier
' diffMm : 1 pour tenir compte de la casse
'          0 pour ne pas tenir compte de la casse
Sub QuickSort(myList(), diffMm As Long)
qSort(myList(), LBound(myList()), UBound(myList()), diffMm)
End Sub
```

Vous trouverez le codage complet dans le fichier du Zip téléchargeable. Le tri fonctionne par récursion sur la routine qSort, qui appelle la fonction partition. Cette dernière compare les éléments deux à deux, ici avec deux appels à la fonction Basic StrComp, et effectue la permutation de deux éléments avec une variable intermédiaire swapping.

La routine serait similaire pour trier une structure logicielle quelconque : le cœur du mécanisme de tri est dans le comparateur d'éléments de la fonction `partition`.

Dans cet exemple de tri, le tableau de `Variant` sera en pratique un tableau de `String`.

```
rem CodeAnnexeB-01.odt   bibli : Trier Module1
Option Explicit

Sub exempleDeTri()
Dim resu As String, bib As String, biblisMacros() As String
biblisMacros = ThisComponent.BasicLibraries.ElementNames
' trier en ordre alphabétique en tenant compte de la casse
QuickSort(biblisMacros(), 1)
resu = join(biblisMacros(), chr(13)) ' convertir en une chaîne
MsgBox(resu, 0, "Bibliothèques Basic de ce document")
End Sub
```

Pour de grands tableaux, l'algorithme de Tri rapide est plusieurs fois plus rapide que l'algorithme de Tri Shell, publié précédemment sur le site fr.OpenOffice.org.

Rappel des routines utilitaires décrites dans le livre

Il nous a semblé utile de récapituler les principales routines réutilisables que nous avons rencontrées au fil des chapitres du livre.

Dialogue

Le chapitre 11 décrit deux routines :

- La fonction `CreerDialogue` 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.

Base de données, formulaires

Les routines `ConnecterSource`, `DeconnecterSource` sont décrites au chapitre 12, ainsi que les fonctions de transformation `Apos` et `PointDec` pour écrire des commandes SQL bien formées.

Les fonctions suivantes se trouvent dans le fichier `CalcSQL.ods` qui se trouve dans le répertoire regroupant les exemples du chapitre 12.

- 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 dans une base de données. Le résultat est obtenu dans un tableau de cellules d'une feuille du tableur.

La fonction `FindCtrlShapeByName`, décrite dans le chapitre 13, recherche dans une page de dessin une forme correspondant à un contrôle de formulaire dont le nom est donné en argument.

Création et décompression d'un fichier Zip

Le chapitre 14 décrit une bibliothèque de gestion de fichier Zip.

Envoyer une commande au Dispatcher

La routine `DispatchSimple`, décrite dans le chapitre 14, permet d'exécuter des commandes de Dispatcher qui ne nécessitent pas d'argument.

Conclusion

Les routines 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 récrire plusieurs fois le même code pour effectuer la même action. Créez vous-même de telles routines ; elles constitueront petit à petit 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.