L’API OpenOffice.org (presque) sans peine

Distribué par le projet fr.OpenOffice.org

Sommaire

1 Introduction

OpenOffice dispose d'un puissant langage de macro, Basic, qui est fourni avec le produit. En plus de ses possibilités classiques de programmation, il permet de manipuler le contenu des documents Writer, Calc, etc. Ceci implique d'utiliser l'interface de programmation (API) de OpenOffice.

Le but du présent document est de montrer comment utiliser l'API avec des macros Basic, pour créer et modifier des documents OpenOffice. Les exemples donnés dans ce document (et de multiples variantes) ont été testés par l'auteur.

L'API est « orienté objet ». La connaissance de la programmation objet vous aidera à comprendre, mais je ne chercherai pas à rentrer dans ce sujet. Ce document s'adresse à un « petit utilisateur » qui n'est pas un programmeur professionnel, et qui cherche des réponses simples à ses besoins.

J'ai utilisé les tournures simplifiées qu'offre Basic, ce qui m'éloigne un peu des tournures valables en Java et autres langages.

1.1 Connaissances préalables

Si vous n'avez pas d'expérience, familiarisez-vous avec l'interface utilisateur et les mécanismes généraux. Pour vous y aider, lisez le HowTo « Débuter – Comment faire et utiliser des macros en Basic », disponible gratuitement sur la page des HowTo, section *Programmation Basic*, du site web fr.OpenOffice.org. Les chapitres « Comment Trouver des Informations sur les Macros » et « D'autres Exemples » sont actuellement (Novembre 2005) obsolètes, mais le reste du document est très didactique.

Vous devriez aussi avoir des connaissances de base en programmation et avoir parcouru une documentation sur le langage OpenOffice Basic lui-même. Vous en trouverez un survol assez complet dans les premiers chapitres du **Guide du Programmeur Staroffice 7**, de Sun MicroSystems. Il est disponible gratuitement via un lien sur la page des HowTo, section *Programmation Basic*, du site web fr.OpenOffice.org. Plutôt que de poursuivre la lecture avec les chapitres « Introduction à l'API OpenOffice.org » et suivants, je conseille de lire maintenant ce HowTo, quitte à revenir plus tard au document Sun.

1.2 Comment aborder ce document

Commencez par lire les chapitres 2 à 4 de ce HowTo. Si vous êtes principalement intéressé par les documents Writer, ou Draw, etc, sautez ensuite à ces chapitres. Eventuellement un lien hypertexte vous renverra à un autre chapitre qui développe un sujet particulier. Utilisez le Navigateur pour sauter au sous-chapitre qui vous intéresse.
Si vous lisez l'anglais, le chapitre Consulter l'API vous permettra d'approfondir un sujet.

N'hésitez pas à réaliser vous-même des petites macros d'essais mettant en oeuvre les notions indiquées. Utilisez abondamment le copier/coller pour recopier des séquences d'instructions depuis ce HowTo ou depuis une macro existante. Utilisez l'aide en ligne pour vérifier la syntaxe de vos instructions Basic : le curseur étant sur l'instruction, taper F1.

1.3 Autres sources d'information

1.3.a Livre sur la programmation OpenOffice.org

Le présent document, malgré ses 100 pages, n'est qu'une introduction à la programmation sur OpenOffice.org. Laurent Godard et moi-même avons écrit un livre bien plus complet : Programmation OpenOffice.org, Macros OOoBasic et API édité chez Eyrolles. Il décrit progressivement la programmation Basic et une grande partie de l'immense API ; il corrige des erreurs des documentations officielles, explique de nombreux points non documentés, et fournit

de multiples exemples de macros exécutables immédiatement. Cet ouvrage fera gagner un temps précieux au programmeur professionnel comme à l'utilisateur exigeant.

1.3.b Autres documentations sur Internet

Le guide « Elements de programmation des macros » de Andrew Pitonyak est disponible en traduction française sur cette page du site web fr.OpenOffice.org. Ce document n'est vraiment accessible qu'à des personnes familiarisées avec l'API. Préférez la version anglaise, souvent mise à jour sur le site de son auteur.

De nombreux documents français concernant la programmation OpenOffice sont téléchargeables gratuitement sur la page des HowTo, section *Programmation Basic*, du site web fr.OpenOffice.org.

1.3.c Aide collégiale

Les discussions en français sur la programmation OpenOffice.org se déroulent sur la liste de diffusion prog@fr.openoffice.org , pour s'inscrire voir la page http://fr.openoffice.org/contact-forums.html

Il existe aussi un forum web très actif : http://www.forum-openoffice.org/

En langue anglaise, le forum web « OpenOffice.org Macros and API » est le plus actif, voir la page http://www.oooforum.org/

1.4 Enregistreur de macro

A partir de la version 1.1 de OpenOffice, un enregistreur de macro est disponible. Il vous permet de répéter une séquence de manipulations effectuées avec l'interface utilisateur de Writer ou Calc (mais pas Draw, ni Impress, ni Base).

Notez que les macros obtenues ne fonctionnent pas dans les versions 1.0.x de OpenOffice.

Le code généré par l'enregistreur ne vous aidera pas à créer des macros vous-même. Il utilise un interface particulière (la fonction dispatch) qui sert à manipuler l'interface utilisateur, tout à fait adapté au but (répéter une séquence utilisateur) mais assez lourde. De plus le code généré n'est pas optimisé. Pour tout projet autre qu'une simple séquence de touches, il est nécessaire de programmer en utilisant Basic et l'API OpenOffice.org.

Après avoir acquis une certaine connaissance du langage macro et de l'API, l'enregistreur de macro pourra vous aider à effectuer des fonctions réalisables par l'interface utilisateur, dont vous ne trouvez pas l'équivalent API.

1.5 Evolution du document

Principales évolutions depuis la version du 3 Novembre 2005

1.5.a Modifications, corrections

1.....Mise à jour des liens internet

6.6.....Correction du codage concernant `DecimalChar` et `FillChar`.

10.2.....Signalement de la macro d'Andrew Brown pour compter les mots d'une sélection, et de l'introduction de la fonctionnalité sur la version 2.

1.5.b Ajouts

2 Caractéristiques du Basic OOo

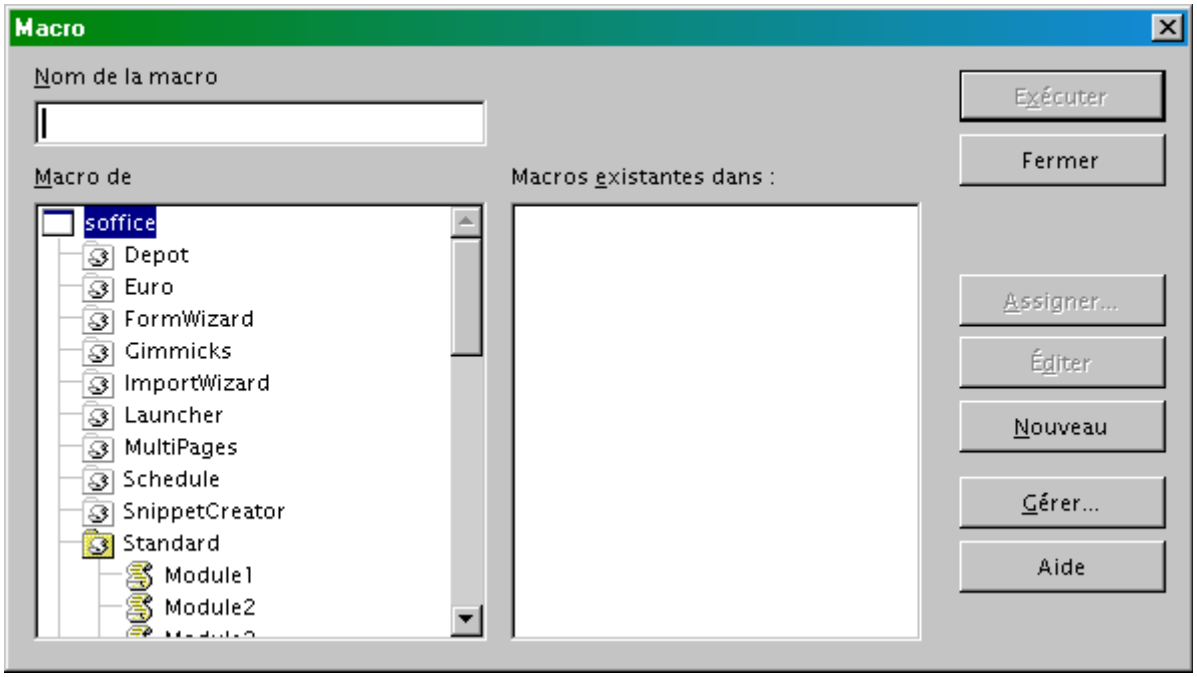
2.1 Accès aux macros

Le HowTo « Comment faire et utiliser des macros en Basic » explique clairement comment créer et éditer des macros.

Entre les versions 1.x et 2.x d'OpenOffice.org les méthodes de programmation et les ressources de l'API sont identiques, avec compatibilité ascendante. Mais l'interface utilisateur a été remaniée.

2.1.a OpenOffice version 1.x

Par le menu Outils > Macros... vous obtenez un panneau comportant une arborescence sur la partie gauche. Les feuilles de l'arborescence sont des modules. Chaque module peut contenir plusieurs macros.



L'arbre *soffice* comporte plusieurs bibliothèques (anglais: libraries), elles-mêmes composées de modules. Il existe de nombreuses macros pré-définies dans cet arbre. Leur lecture est ardue, mais instructive. La bibliothèque *Standard* de *soffice* reçoit les macros qui seront accessibles directement par tous les documents OpenOffice.

Chaque document ouvert possède un arbre à son nom, avec une bibliothèque *Standard*. La bibliothèque *Standard* d'un document reçoit les macros directement accessibles dans ce document. Chaque module fait l'objet d'un document xml intégré dans le fichier du document.

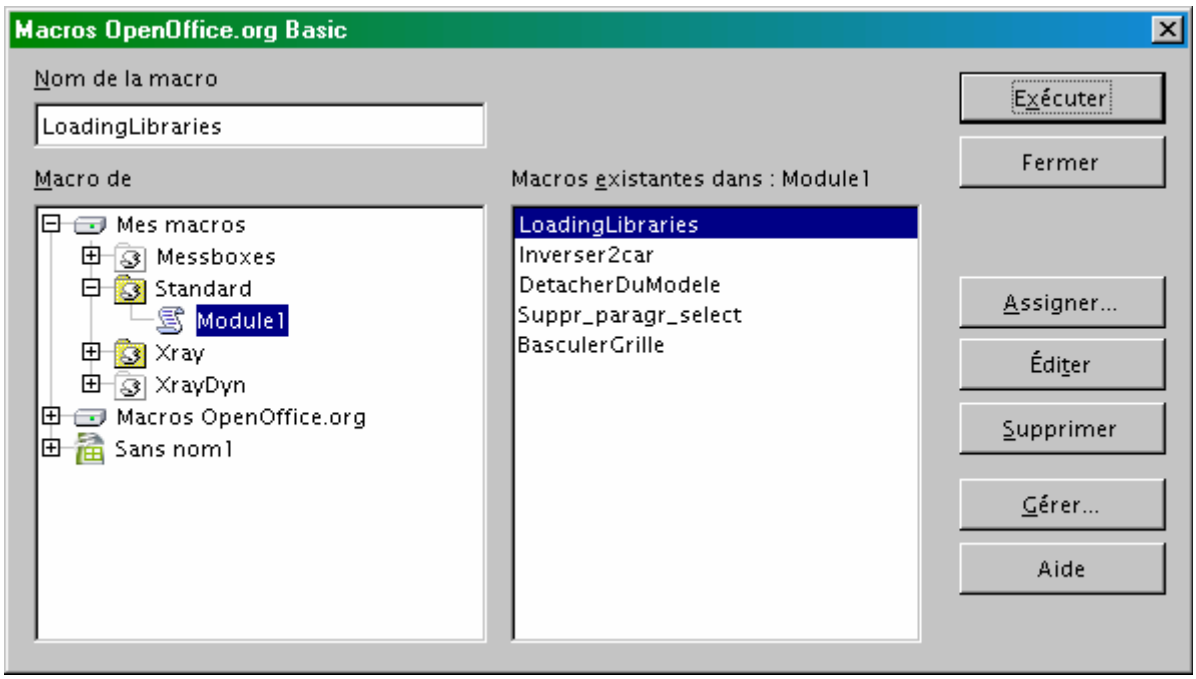
Si plusieurs documents sont ouverts en même temps, l'arborescence vous donne accès aux macros de tous les documents ! Faites très attention à définir votre macro dans le bon document !

Pour écrire une macro utilisez l'éditeur de macro intégré à OpenOffice.

Dans l'éditeur de macros vous bénéficiez d'une coloration syntaxique très utile, que je ne reproduit pas ici. Je vous conseille de copier/coller les exemples dans l'éditeur de macros et de les exécuter en pas à pas.

2.1.b OpenOffice version 2.x

L'accès et le contenu sont un peu plus complexes. Par le menu Outils > Macros > Gestionnaire de Macros > OpenOffice.org Basic... vous obtenez un panneau comportant une arborescence sur la partie gauche. Les feuilles de l'arborescence sont des modules. Chaque module peut contenir plusieurs macros.



À la différence des versions 1.x d'OpenOffice vous avez deux racines, en plus de celles correspondant à chaque document.

- La racine **Mes Macros** contient les macros que vous avez créées ou importées et qui sont disponibles pour tous les documents.
- La racine **Macros OpenOffice.org** contient les macros livrées avec OpenOffice. En principe vous ne pouvez pas les modifier, mais vous pouvez les lire ou les utiliser.

En réalité ces deux racines ne sont là que pour vos yeux. Toutes les deux sont un seul et même conteneur, qu'on appelle `soffice` en version 1.x. Ce terme est actuellement encore utilisé dans certaines documentations de la version 2 ainsi que dans le titre par défaut des panneaux `MsgBox`.

Fonctionnellement, le reste de la description est identique à celle de la version 1.x.

2.2 Esthétique et lisibilité

Ce document utilise des styles spécifiques pour les instructions de macro :

- `style de paragraphe : ligneMacro`
- style de caractère : `InstructionMacro`
- style de caractère : `InstructionMacroChangement`
- style de caractère : *NomdeVariable*

Adaptez éventuellement ces styles si vous avez des difficultés à lire ou imprimer, tout le document suivra.

La lecture de ce document sur écran permet de sauter directement au sujet qui vous intéresse

grâce au navigateur d'OpenOffice et aux liens hypertexte qui se trouvent dans la table des matières et dans le corps du document.

2.3 Eléments syntaxiques du Basic OpenOffice.org

Basic OOo ne fait pas la distinction entre lettres majuscules et minuscules. Y compris pour les noms de variables et les noms réservés. Mais en utilisant majuscule et minuscule, c'est plus lisible. Par contre certains éléments de l'API doivent être écrits avec les majuscules et minuscules indiquées; on le signalera dans ce document.

Vous pouvez insérer un nombre quelconque d'espaces entre deux mots Basic. Les lignes vides sont aussi ignorées par Basic.

Un commentaire commence avec `Rem` (pas obligatoirement en début de ligne) et se termine à la fin de la ligne. Une apostrophe simple `'` est équivalente à un `Rem`.

2.4 Définitions de variables

2.4.a Conseils

Insérez au tout début de votre module de macro l'instruction

```
Option Explicit
```

Grâce à cette instruction, vous aurez un message d'erreur pour toute variable non définie, ou mal orthographiée.

Définissez chaque variable en précisant son type explicitement. Donnez un nom significatif à vos variables importantes. C'est un tout petit peu plus long à écrire, mais bien plus clair, et vous réduirez le temps passé à chercher les erreurs de programmation.

Exemple

```
OPTION EXPLICIT

' variables communes au module entier
Dim MonDocument  As Object,  LesFeuilles  As Object
Dim MaFeuille As Object

Sub LesCompositeurs

Dim cellIndex As Object, cellEgale As Object ' variables locales
Dim y1 As Integer, x2 As Integer ' variables locales au sous-programme
rem  -- ici les instructions de la subroutine --
End Sub
```

Note :

Si vous déclarez plusieurs variables avec le même Dim, répétez le type pour chacune. Sinon les variables sans type sont implicitement des Variant (le type est défini à l'exécution).

```
Dim nombre3, nombre4 As Long   rem nombre3 est un Variant
```

Ceci peut avoir des effets bizarres, un Variant étant initialisé à « Empty ».

2.4.b Choix des noms dans ce document

Dans les exemples fournis, j'utiliserai en général des variables avec un nom français pour bien les distinguer des primitives du langage Basic et démystifier l'aspect « formule magique ». Je trouve cela plus clair que la méthode de codage employée dans le guide Basic qui consiste à utiliser un nom anglais précédé d'une lettre préfixe (oCell pour : objet Cellule).

3 Trouver le document

On initialise une variable objet pour la faire pointer sur le document à manipuler.

3.1 Le document en cours

En général le document sur lequel la macro va travailler est le document dont la fenêtre est en avant-plan (elle a le « focus »). L'initialisation se fait ainsi :

```
Dim MonDocument As Object

MonDocument = ThisComponent
```

3.2 Accèder à un autre document existant

Pour ouvrir un document existant :

```
Dim MonDocument As Object
Dim AdresseDoc As String
Dim PropFich()

AdresseDoc = "file:///home/testuser//work/undocument.sxc"

MonDocument = StarDesktop.LoadComponentFromURL(AdresseDoc,"_blank",0, PropFich)
```

Vous pouvez utiliser la plupart des lignes sans comprendre. Vous avez seulement à adapter la ligne qui définit *AdresseDoc*. Cette adresse est sous forme d'une URL. Pour les utilisateurs Windows voici l'équivalence :

```
C:\Mes Documents\tata.sxc          file:///C|/Mes%20Documents/tata.sxc
```

Basic fournit une commande pour convertir une adresse Windows en URL, exemple :

```
AdresseDoc = convertToURL("C:\Documents and Settings\Arthur\Mes documents\Ma Petite
Doc.sxw")
```

Notez que l'URL peut aussi désigner une adresse sur le réseau local.

3.3 Créer un nouveau document

La méthode est celle du chapitre précédent, mais on utilise une adresse magique, qui dépend du type de document ; exemple pour un document Calc :

```
Dim MonDocument As Object
Dim AdresseDoc As String
Dim PropFich()

AdresseDoc = "private:factory/scalc"
MonDocument = StarDesktop.LoadComponentFromURL(AdresseDoc,"_blank",0, PropFich)
```

Type de nouveau document	*Adresse magique*
Texte Writer	`private:factory/swriter`
Tableur Calc	`private:factory/scalc`
Dessin Draw	`private:factory/sdraw`
Présentation Impress	`private:factory/simpress`
Editeur de formules Math	`private:factory/smath`

Ces documents suivent le modèle par défaut en vigueur pour votre configuration de OpenOffice.

3.4 Créer un nouveau document à partir d'un modèle

Pour créer un nouveau document conforme à un modèle particulier, mettre dans *AdresseDoc* le chemin d'accès au modèle.

```
AdresseDoc = convertToURL("C:\Mes modeles\Doc2colonnes.stw")
```

C'est très pratique pour disposer immédiatement d'un ensemble de styles à votre goût, de feuilles Calc prédéfinies, d'un en-tête ou un pied de page, d'une mise en page en colonnes, ou d'un texte initial qui sera complété par la macro.

3.5 Quelques détails complémentaires

3.5.a StarDesktop

L'objet `StarDesktop` simplifie les choses. Sans lui on écrirait :

```
Dim MonBureau As Object, MonDocument As Object
Dim AdresseDoc As String
Dim PropFich()

MonBureau = createUnoService("com.sun.star.frame.Desktop")

AdresseDoc = "private:factory/scalc"
MonDocument = MonBureau.LoadComponentFromURL(AdresseDoc,"_blank",0, PropFich)
```

3.5.b ThisComponent

L'instruction Basic `ThisComponent` simplifie les choses. Mémorisez *en début d'exécution* de votre macro la valeur de `ThisComponent`

```
MonDocument = ThisComponent
```

Pourquoi utiliser une variable intermédiaire ? Parce que `ThisComponent` n'est pas un objet fixe, c'est une sorte de sous-programme qui renvoie l'objet document actuel. Le résultat dépendra des conditions règnant au moment exact où vous appelez `ThisComponent`.

Si vous n'utilisez pas une variable intermédiaire, et que votre macro dure un certain temps, l'utilisateur peut mettre en avant-plan un autre document OpenOffice et vos appels successifs de `ThisComponent` donneront des résultats différents.

Si vous appelez `ThisComponent` dans un sous-programme situé dans `soffice`, il renvoie l'objet document OpenOffice actuellement en avant-plan.

Si vous l'appelez depuis un sous-programme *situé dans un document*, il renvoie cet objet document ; même si vous avez appelé ce sous-programme depuis un autre document ouvert, avec le menu Outils > Macros.

On voit souvent la séquence suivante (francisée) :

```
MonBureau = createUnoService("com.sun.star.frame.Desktop")
MonDocument = MonBureau.getCurrentComponent()
MonTexte = MonDocument.Text
```

Ceci **n'est pas** l'équivalent de

```
MonDocument = ThisComponent
MonTexte = MonDocument.Text
```

mais de :

```
MonDocument = StarDesktop.CurrentComponent
MonTexte = MonDocument.Text
```

Si vous exécutez la première séquence de macro dans l'EDI (la fenêtre d'affichage des macros), la troisième instruction va tomber en faute : *propriété ou méthode introuvable*. Parce que `getCurrentComponent` renvoie la fenêtre de l'EDI, et non pas le document sur lequel la macro s'exerce.

La deuxième séquence fonctionne aussi dans l'EDI, preuve que `ThisComponent` est plus malin. Si vous voulez exécuter vos macros dans l'EDI, et profiter de ses facilités de débogage, utilisez donc `ThisComponent`.

3.5.c Arguments de LoadComponentFromURL

`LoadComponentFromURL` sert à charger un document; il sera affiché dans un cadre (une fenêtre).

- Le deuxième argument de `LoadComponentFromURL` est le nom du cadre cible. Si un cadre (une fenêtre) ayant ce nom existe, le document sera affiché dedans. Il existe certains noms réservés (voir API), mais vous n'en aurez probablement pas besoin.
- Le troisième argument de `LoadComponentFromURL` est en général inutilisé (valeur zéro).
- Le quatrième argument contient les propriétés souhaitées du fichier, voir ci-dessous.

3.5.d Propriétés pour ouvrir un document

Vous pouvez préciser certaines informations, détaillées dans l'API, voir le service `MediaDescriptor`. Chaque information possède une structure de type Propriété; chaque propriété a un nom (Name) et une valeur (Value). Si vous n'avez pas d'information particulière à fournir, vous devez au moins fournir un élément vide, comme indiqué plus haut.

Comme exemple, ouverture d'un document protégé par un mot de passe :

```
Dim MonDocument As Object
Dim AdresseDoc As String
Dim PropFich(0) As New com.sun.star.beans.PropertyValue

AdresseDoc = "file:///home/testuser//work/Finances.sxc"

PropFich(0).Name = "Password"
PropFich(0).Value = "Arthur1987"
MonDocument = StarDesktop.LoadComponentFromURL(AdresseDoc,"_blank",0, PropFich() )
```

Respectez les majuscules-minuscules pour le nom de la propriété. Remarquez aussi que nous définissons *PropFich* comme un vecteur à un élément, et que les parenthèses deviennent obligatoires dans la dernière ligne. Pour 3 propriétés simultanées, *PropFich* serait défini comme :

```
Dim PropFich(2) As New com.sun.star.beans.PropertyValue
```

Propriétés principales pour ouvrir un document

Propriété	***Signification***
`AsTemplate`	booléen; utile si on travaille sur un modèle. valeur true : (par défaut) un nouveau document sans titre est créé à partir du modèle désigné dans l'URL ; valeur false : le modèle est ouvert pour édition. Remarque Avec `AsTemplate = true` et une *AdresseDoc* correspondant à un document ordinaire (pas un modèle) on obtient un nouveau document basé sur celui en référence.

Propriété	*Signification*
`Hidden`	booleen ; valeur true pour charger le document sans le rendre visible
`Password`	chaîne de caractères ; mot de passe pour ouvrir le document encrypté
`ReadOnly`	booléen ; valeur true pour ouvrir le document en lecture seule (pour l'usager, pas pour l'API)

4 Sauver le document

D'abord, le document a-t-il été modifié ? Ce test le détermine :

```
if MonDocument.isModified then
  rem mettre ici les instructions pour sauver le document
end if
```

Si vous avez modifié un document existant, le sauver est tout simple :

```
MonDocument.Store
```

Si vous avez ouvert un nouveau document, vous devez préciser sous quel nom et à quelle adresse il doit être sauvé; on indique pour cela une URL.

Vous pouvez aussi préciser certaines informations, détaillées dans l'API, voir le service `MediaDescriptor`. Chaque information possède une structure de type Propriété; chaque propriété a un nom (Name) et une valeur (Value). Si vous n'avez pas d'information particulière à fournir, vous devez au moins fournir un élément vide.

Cet exemple écrase tout document du même nom à la même adresse :

```
Dim MonDocument As Object
Dim NomduFichier As String
Dim PropFich()

NomduFichier = "file:///C:/Travail/__blabla.sxw"
MonDocument = ThisComponent

MonDocument.storeAsURL(NomduFichier, PropFich)
```

Pour ne pas écraser un document existant, on utilisera la propriété `Overwrite` :

```
Dim MonDocument As Object
Dim NomduFichier As String
Dim PropFich(0) As New com.sun.star.beans.PropertyValue

NomduFichier = "file:///C:/Travail/__blabla.sxw"
MonDocument = ThisComponent
PropFich(0).Name = "Overwrite"
PropFich(0).Value = false
MonDocument.storeAsURL(NomduFichier, PropFich() )
```

Il y aura une exception (à traiter par programmation) si le document existe déjà.

Respectez les majuscules-minuscules pour le nom de la propriété. Remarquez aussi que nous définissons *PropFich* comme un vecteur à un élément, et que les parenthèses deviennent obligatoires dans la dernière ligne. Pour 3 propriétés simultanées, *PropFich* serait défini comme :

```
Dim PropFich(2) As New com.sun.star.beans.PropertyValue
```

Propriétés principales pour enregistrer le document

Propriété	***Signification***
`Author`	une chaîne de caractères pour satisfaire votre ego
`Version`	un nombre entier
`Password`	chaîne de caractères ; mot de passe pour ouvrir le document encrypté

4.1 Enregistrer sous / Enregistrer une copie sous

La méthode `storeAsURL` réalise l'équivalent du menu Fichier > Enregistrer sous...

Si vous avez ouvert un fichier TOTO et que vous l'enregistrez sous TATA, vous travaillez

maintenant dans le document TATA.

Par contre, la méthode `storeToURL` réalise une copie du document en cours, et seulement cela. Ainsi, travaillant sur TOTO, vous faites une copie appelée TATA et continuez sur TOTO. Utile pour des sauvegardes régulières.

La méthode `storeToURL` utilise la même syntaxe que `storeAsURL`.

4.2 Fermer le document

La méthode `Close` ferme le document, et donc la fenêtre qui l'affichait. Exemple :

```
Dim UnDocument As Object
Dim AdresseDoc As String

AdresseDoc = "file:///C:/Textes/blabla.sxw"
UnDocument = StarDesktop.LoadComponentFromURL(AdresseDoc,"_blank",0, PropFich)
print "document ouvert !"
UnDocument.Close(True)
```

Dans tous les cas pratiques, il faut mettre en argument de `Close` la valeur booleenne `True`.

Si OpenOffice n'a plus d'autre fenêtre active, et si le Démarrage rapide n'est pas utilisé, l'application OpenOffice.org se ferme.

5 Tableur Calc

5.1 Trouver la feuille

Avant de travailler sur des cellules, vous devez préciser dans quel document elles se trouvent, et dans quelle feuille.

5.1.a Une feuille existante

Chaque feuille de Calc comporte un onglet. Le nom des onglets est par défaut, dans la version localisée française, Feuille1, Feuille2, etc.

Avec l'API on accède ainsi à une feuille :

```
Dim MonDocument As Object
Dim MaFeuille As Object, LesFeuilles As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
print "Nombre de feuilles : "; LesFeuilles.Count
MaFeuille = LesFeuilles(0) ' Première feuille
```

`Sheets` fournit l'ensemble des feuilles du document. `Count` donne le nombre de feuilles existantes. Chaque feuille est accessible par un index.

On aurait pu se passer de la variable *LesFeuilles* en écrivant la dernière ligne :

```
MaFeuille = MonDocument.Sheets(0) ' Première feuille du tableur
```

Attention

Pour l'API les pages sont numérotées à partir de zéro, dans l'ordre affiché par l'interface Calc. Ainsi si l'utilisateur change l'ordre des feuilles, la macro écrira obstinément sur la feuille de rang n.

Une meilleure méthode consiste à récupérer la feuille ayant un nom donné :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("Oeuvres")
```

On peut tester si une feuille d'un nom donné existe :

```
if LesFeuilles.hasByName("Oeuvres") then
  rem  la feuille existe bien
end if
```

5.1.b Créer, renommer, dupliquer, supprimer une feuille

Créer une feuille vierge

La méthode `insertNewByName` insère une nouvelle feuille vierge à une position donnée parmi les feuilles du document, et lui donne un nom.

```
Dim MonDocument As Object
Dim NouvelleFeuille As Object, LesFeuilles As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
' insérer la feuille vierge XXX après la deuxième feuille
LesFeuilles.insertNewByName("Totalisation", 2)
NouvelleFeuille = LesFeuilles.getByName("Totalisation")
```

Renommer une feuille

Renommer une feuille revient à changer la valeur de la propriété `Name` de la feuille.

```
MaFeuille = LesFeuilles.getByName("Mois")
MaFeuille.Name = "Fevrier"
```

Déplacer une feuille

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("Fevrier")
' Déplacer en deuxième position
LesFeuilles.MoveByName("Fevrier", 1)
```

Dupliquer une feuille

```
Dim MonDocument As Object, LesFeuilles As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
' Créer une copie en troisième position
LesFeuilles.CopyByName("Fevrier", "Mars", 2)
```

La nouvelle feuille Mars est la copie exacte de la feuille Fevrier, avec son contenu.

Supprimer une feuille

```
Dim MonDocument As Object, LesFeuilles As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
LesFeuilles.removeByName("Fevrier")
```

5.1.c La feuille visible par l'utilisateur

La feuille visible par l'utilisateur se trouve ainsi :

```
MaFeuille = MonDocument.CurrentController.ActiveSheet
print "Le nom de la feuille active est : "; MaFeuille.Name
```

Et pour rendre visible une autre feuille :

```
MaFeuille = MonDocument.Sheets(1)
MonDocument.CurrentController.ActiveSheet = MaFeuille
```

5.2 Trouver la cellule

Pour manipuler une cellule, je vais définir une variable intermédiaire qui pointera sur la cellule.

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object, UneCellule As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("Oeuvres")
UneCellule = MaFeuille.getCellByPosition(3,5) ' cellule D6
```

Les paramètres de `getCellByPosition(x,y)` sont :

- x = rang de la colonne, avec 0 pour A, 1 pour B, 25 pour Z, etc
- y = rang de la ligne, avec 0 pour la ligne 1, 1 pour la ligne 2, etc.

Nous retrouverons cette méthode de positionnement dans d'autres fonctions.

5.3 Zone de cellules

5.3.a Manipuler une zone de cellules

Il existe un type d'objet « zone de cellules ». On définit une zone de cellules de différentes manières :

```
Dim UneZone As Object

Rem zone B1 à C6 dans la 4 ème feuille du document Calc
  UneZone =  GetCellRange(MonDocument, 3, 1, 0, 2, 5)
Rem zone dans une feuille particulière
  UneZone =  MaFeuille.getCellRangeByName("B1:C6")
  UneZone =  MaFeuille.getCellRangeByName("$B2")  ' zone 1 cellule
  UneZone =  MaFeuille.getCellRangeByName("NomdeZone")
  UneZone =  MaFeuille.getCellRangeByPosition(1,0,2,5) ' zone B1:C6
```

Pointer sur une cellule dans la zone :

```
UneCellule = UneZone.getCellByPosition(3,5)
```

ici les coordonnées, toujours numérotées à partir de zéro, sont relatives au coin haut-gauche de la zone.

Si la zone de cellule se réduit à une seule cellule, on a en fait un pointeur vers une cellule :

```
UneCellule =  MaFeuille.getCellRangeByName("C13")
```

Remarque :

Nous avons vu deux utilisations de `getCellByPosition` :

```
1. MaFeuille.getCellByPosition(3,5)
2. UneZone.getCellByPosition(3,5)
```

Il est utile de savoir qu'il ne s'agit pas de la même fonction, mais de deux fonctions similaires.

Dans le premier cas, la fonction est spécifique à un objet de type feuille, dans le second cas la fonction est spécifique à un objet de type zone de cellule. Ce concept est typique de la conception orientée objet : chaque type d'objet possède des variables, des procédures et des fonctions qui lui sont propres; il est d'usage de reprendre le même nom entre deux types d'objets, quand les fonctions sont similaires. Par conséquent, pour lever l'ambigüité on qualifie le nom de la fonction ou procédure ou variable, avant le nom de l'objet.

Si vous parcourez un jour la documentation de l'API vous trouverez nombre de fonctions et procédures synonymes. Vous ne pouvez pas les utiliser sans savoir à quel type d'objet elles appartiennent.

5.3.b Zone de cellules sélectionnées par l'utilisateur

L'utilisateur peut sélectionner une zone de cellules, puis appeler une macro. La séquence suivante retrouve la zone sélectionnée.

```
Dim MonDocument As Object
Dim UneZone As Object

MonDocument = ThisComponent
UneZone = MonDocument.CurrentSelection
```

L'utilisateur pourrait sélectionner plusieurs zones à la fois, au lieu d'une seule zone. Dans ce cas, `CurrentSelection` fournit une liste des zones sélectionnées. Nous pouvons le savoir ainsi :

```
Dim MonDocument As Object
Dim UneZone As Object, LesSelections As Object
Dim selx As Integer

MonDocument = ThisComponent
```

```
LesSelections = MonDocument.CurrentSelection
if LesSelections.supportsService("com.sun.star.sheet.SheetCellRanges") then
  for selx = 0 to LesSelections.Count -1
  UneZone = LesSelections(selx)
    ' ici, utiliser une des zones sélectionnées
  next
else
  ' une seule zone ou une cellule
end if
```

On utilise alors une boucle pour trouver successivement ces zones. Le nombre de zones est dans la propriété `Count`.

5.3.c Sélectionner visiblement une zone de cellules

Le but est de sélectionner une zone par macro, et d'afficher cette sélection à l'utilisateur. Il faut une instruction spéciale, qui utilise en argument un pointeur vers une cellule ou une zone de cellules :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object, UneZone As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("Feuille3")
UneZone =  MaFeuille.getCellRangeByName("B2:F6")
MonDocument.CurrentController.Select(UneZone)
```

Si vous avez exécuté cette macro dans l'EDI, mettez en avant-plan la fenêtre Calc pour voir la sélection du mot.

Notez aussi que la feuille concernée est automatiquement affichée.

5.4 Retrouver les coordonnées d'une cellule

Maintenant que nous savons obtenir une zone de cellules, et rechercher une cellule dans la zone, il est parfois utile de connaître les coordonnées absolues d'une zone ou d'une cellule.

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object, UneZone As Object, UneCellule As Object
Dim CoordCellule As Object, CoordZone As Object
Dim FeuilleCourante As Integer, ColonneCourante As Integer
Dim LigneCourante As Integer, FeuilledeZone As Integer
Dim PremiereColonneZone As Integer, DerniereColonneZone As Integer
Dim PremiereLigneZone As Integer, DerniereLigneZone As Integer

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("Feuille3")
UneZone =  MaFeuille.getCellRangeByName("B1:G6")
UneCellule = UneZone.getCellByPosition(2,4) ' position dans la zone

CoordCellule = UneCellule.CellAddress
FeuilleCourante = CoordCellule.Sheet
ColonneCourante = CoordCellule.Column
LigneCourante = CoordCellule.Row

CoordZone = UneZone.RangeAddress
FeuilledeZone = CoordZone.Sheet
PremiereColonneZone = CoordZone.StartColumn
DerniereColonneZone = CoordZone.EndColumn
PremiereLigneZone = CoordZone.StartRow
DerniereLigneZone = CoordZone.EndRow
```

Remarquez la similarité des instructions pour une zone et pour une cellule.

Ici encore, les coordonnées obtenues sont numérotées à partir de zéro.

5.5 Connaître le type de contenu de la cellule

En supposant avoir écrit les lignes des chapitres précédents, on peut maintenant lire et écrire dans la cellule. Mais les instructions dépendent du genre d'information à manipuler.

Une cellule a un parmi quatre types possibles de contenu :

- cellule vide
- valeur numérique
- un texte
- une formule(qui peut fournir une valeur numérique ou un texte)

`Type` fournit le type de contenu de la cellule , sous forme d'une valeur numérique.

Pour être indépendant de l'implémentation (qui peut évoluer), chaque type doit être désigné avec une constante prédéfinie, dont j'explique le mode d'emploi au chapitre Les horribles constantes.

Cet exemple liste les constantes et montre comment les utiliser pour connaître le contenu d'une cellule :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object, UneCellule As Object
Dim ContTexte As Integer, ContValeur As Integer
Dim ContFormule As Integer, ContVide As Integer

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("Feuille2")
UneCellule =MaFeuille.getCellByPosition(2,9) ' cellule C10

rem   constantes des types de contenu
ContVide=    com.sun.star.table.CellContentType.EMPTY
ContValeur=  com.sun.star.table.CellContentType.VALUE
ContTexte=   com.sun.star.table.CellContentType.TEXT
ContFormule= com.sun.star.table.CellContentType.FORMULA

Select Case UneCellule.Type
Case ContVide
  Print "case vide !"
Case ContFormule
  Print "Formule"
Case ContValeur
  Print "Valeur"
Case ContTexte
  Print "Texte"
End Select
```

5.6 Valeur numérique dans la cellule

Lire et écrire une valeur numérique dans la cellule :

```
Dim x2 As Integer
x2 = UneCellule.Value
x2 = x2 + 17
UneCellule.Value = x2
```

Attention : `Value` vous donne le résultat numérique de la formule qui se trouve dans la cellule. Si la cellule contient un texte, ou si la formule donne un texte, `getValue` donne zéro.

5.7 Texte dans la cellule

Lire et écrire une chaîne de caractère dans la cellule :

```
Dim tx3 As String
tx3 = UneCellule.String
```

```
UneCellule.String = "Bonsoir"
```

La dernière instruction écrase tout texte existant dans la cellule. On peut faire mieux, mais c'est plus compliqué. Cependant le principe se retrouve pour écrire du texte dans Writer ou ailleurs.

5.7.a Le curseur d'écriture de texte dans la cellule

Pour écrire un texte dans un texte existant dans une cellule, nous avons besoin d'un curseur d'écriture. Ce curseur indique le point d'insertion du nouveau texte, et avance à chaque insertion.

La séquence suivante insère un texte à droite de celui existant dans la cellule :

```
Dim MonCurseur As Object

tx3 = "Bonjour"
MonCurseur = uneCellule.createTextCursor
UneCellule.insertString(MonCurseur, tx3, false)
UneCellule.insertString(MonCurseur, " les amis", false)
```

Le curseur est positionné à sa création à la fin du texte existant dans la cellule. La gestion du curseur d'écriture est décrite au chapitre suivant.

Dans la procédure `insertString` le troisième argument signifie :

- `false` = insérer dans le texte
- `true` = écraser dans le texte

5.7.b Caractères spéciaux dans un texte

Même dans une simple cellule on peut insérer une fin de paragraphe de texte :

```
Dim FinParagraphe As Integer
FinParagraphe = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
UneCellule.insertControlCharacter(MonCurseur, FinParagraphe, false)
```

Il existe une liste d' horribles constantes similaires :

Constante	*Signification*
`PARAGRAPH_BREAK`	Insérer ici une marque de fin de paragraphe.
`APPEND_PARAGRAPH`	Insérer un paragraphe à la fin de celui en cours et se positionner au début du nouveau paragraphe.

5.7.c Déplacement du curseur de texte dans une cellule

Pour insérer à un endroit quelconque nous avons besoin de modifier la position du curseur.

```
MonCurseur.goLeft(23, false)     ' déplace de 23 caractères vers gauche
MonCurseur.goRight(12, false)    ' déplace de 12 caractères vers droite
MonCurseur.gotoStart(false)        ' vers début du texte
MonCurseur.gotoEnd(false)        ' vers fin du texte
```

Le dernier argument des routines curseur signifie :

- `false` = déplacer le curseur
- `true` = déplacer le curseur en étendant la sélection (comme en glissant avec la souris)

On peut donc sélectionner une zone de plusieurs caractères; ceci fait, on récupère le texte sélectionné ainsi :

```
tx3 = MonCurseur.String
```

Inversement, on peut écraser un texte sélectionné avec un nouveau texte :

```
MonCurseur.String = "Bonsoir"
```

Il existe aussi : `collapseToStart`, `collapseToEnd`, `isCollapsed`, `gotoRange`. Autres informations sur le curseur : voir dans l'API : XTextCursor et XTextRange

5.8 Formule dans la cellule

Lire une formule dans la cellule :

```
Dim tx3 As String
tx3 = UneCellule.Formula 'la formule en version anglaise
tx3 = UneCellule.FormulaLocal 'la formule en version localisée
```

Si votre cellule contient =Maintenant() vous obtiendrez =Now() en version anglaise.

Ecrire une formule dans la cellule, en version localisée :

```
Dim x2 As Integer
x2 = 5
UneCellule.FormulaLocal= "=Arrondi(AutreF.A" + LTrim(Str(x2))+")"
```

La variable numérique x2 est transformée en chaîne de caractère, `LTrim` sert à supprimer l'espace avant le chiffre. Résultat : la formule référence la cellule A5 dans la feuille «AutreF» :

```
=ARRONDI(AutreF.A5)
```

On obtiendrait le même résultat en formulation anglaise :

```
UneCellule.Formula= "=Round(AutreF.A" + LTrim(Str(x2))+")"
```

5.9 Recalculer les formules des cellules

Voici comment mettre à jour les calculs de formules dans toutes les cellules du tableur :

```
MonDocument.calculateAll
```

Dans un tableur très complexe, on peut se limiter à recalculer les formules qui ne sont pas à jour :

```
MonDocument.calculate
```

Le recalcul est normalement automatique. On peut désactiver l'automatisme pour éviter des calculs inutiles, puis le réactiver quand on l'estime utile.

```
rem désactiver le recalcul automatique
MonDocument.enableAutomaticCalculation(false)
   rem - - effectuer divers travaux - -
rem activer le recalcul automatique
MonDocument.enableAutomaticCalculation(true)
```

Enfin, on peut déterminer si l'automatisme est en fonction :

```
if MonDocument.isAutomaticCalculationEnabled then
   rem le recalcul automatique est actuellement activé
end if
```

5.10 Modifier le format d'une cellule

Tout ce que nous avons fait jusqu'à maintenant ne change pas le formatage de la cellule. Le texte ou la valeur s'inscrivent avec le formatage existant.

Il existe de très nombreuses possibilités de formatage, voir dans l'API à CellProperties. Les plus courantes sont décrites ci-après.

5.10.a Style d'une cellule

Le plus rapide pour formater une cellule est d'utiliser un style de cellule qui est déjà défini dans le

tableur. La propriété `CellStyle` est une chaîne de caractères qui est le nom du style. On peut lire le nom du style en cours ou changer de style :

```
print UneCellule.CellStyle
UneCellule.CellStyle= "MonStyleAMoi"
```

Attention

Si vous récupérez le nom du style en cours, pour les styles standards fournis avec OOo vous obtiendrez le nom anglais, même avec une version localisée de OOo. Correspondances pour la version française :

Nom en français	*Nom anglais obtenu*
Résultat	Result
Résultat2	Result2
Standard	Default
Titre	Heading
Titre1	Heading1

Donc, en cas de doute, lisez le nom du style avec une petite macro.

Deuxième problème potentiel : la chaîne de caractère du style que vous affectez à une cellule doit reproduire exactement le nom du style existant : majuscules, minuscules, accents. Sinon le style de la cellule restera inchangé.

Les possibilités de formatage qui suivent sont spécifiques à la cellule concernée, elles ne modifient pas le style de cellule. De plus une interrogation du style de la cellule donnera toujours le même nom, alors que la cellule est en réalité formatée différemment.

Notez aussi que les propriétés de cellule peuvent être appliquées directement à une zone de cellule, par exemple la couleur de fond.

5.10.b Couleur de fond de la cellule

```
UneCellule.CellBackColor = RGB(255,255,204) ' jaune pâle
UneZone.CellBackColor = RGB(255,255,204) ' colorer toute une zone
```

La définition des couleurs est décrite au chapitre Couleurs.

5.10.c Couleur de caractère

```
UneCellule.CharColor = RGB(200,0,0) ' rouge
```

La définition des couleurs est décrite au chapitre Couleurs.

5.10.d Taille de caractère

La taille des caractères est définie en points

```
UneCellule.CharHeight = 12
```

5.10.e Justification horizontale

Encore des horribles constantes...

```
UneCellule.HoriJustify = com.sun.star.table.CellHoriJustify.CENTER
```

Principales valeurs possibles :

Constante	Signification
STANDARD	alignement par défaut selon le type du contenu (texte, nombre)
LEFT	Alignement à gauche
CENTER	Alignement centré
RIGHT	Alignement à droite

5.10.f Justification verticale

Encore des horribles constantes...

```
UneCellule.VertJustify = com.sun.star.table.CellVertJustify.TOP
```

Agrandir la hauteur de la ligne pour mieux voir l'effet. Valeurs possibles :

Constante	Signification
STANDARD	alignement par défaut
TOP	alignement sur le haut de la cellule
CENTER	alignement centré en hauteur
BOTTOM	alignement sur le bas de la cellule

5.10.g Orientation verticale/horizontale

La propriété `Orientation` est prise en compte seulement si la propriété `RotateAngle` vaut zéro.

Encore des horribles constantes...

```
UneCellule.Orientation = com.sun.star.table.CellOrientation.TOPBOTTOM
```

Valeurs possibles :

Constante	Signification
STANDARD	Horizontal, tout simplement
TOPBOTTOM	Pour lire, pencher la tête à droite pour un texte européen
BOTTOMTOP	Pour lire, pencher la tête à gauche pour un texte européen
STACKED	Chaque lettre est horizontale, les lettres sont placées de haut en bas.

5.10.h Orientation à angle quelconque

```
UneCellule.RotateAngle = 15*100
```

La valeur est l'angle en centièmes de degré; l'exemple correspond à un angle de 15°. Une valeur positive correspond au sens trigonométrique (le sens inverse des aiguilles d'une montre).

5.10.i Retour à la ligne

Ce formatage est accessible manuellement par : Formater la cellule > onglet alignement > Enchaînements, Renvoi à la ligne. Un texte trop long pour la largeur de la cellule est « replié » sur plusieurs lignes automatiquement.

Cet exemple de code montre comment tester l'état actuel, et le modifier.

```
if UneCellule.IsTextWrapped then
```

```
 print "Le retour à la ligne est actif"
 UneCellule.IsTextWrapped= false  'désactiver le retour à la ligne'
else
 print "Le retour à la ligne est inactif"
 UneCellule.IsTextWrapped= true  'activer le retour à la ligne'
end if
```

5.11 Formater des zones de cellules

Les possibilités de formatage d'une cellule sont aussi disponibles pour une zone de cellules, et même pour des lignes ou colonnes (voir plus loin), et pour la feuille entière comme dans cet exemple :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles.getByName("BelleFeuille")
MaFeuille.CellBackColor = RGB(255,255,204) ' jaune pâle
MaFeuille.CharColor = RGB(200, 0, 0) ' rouge
```

5.12 Lignes et colonnes

5.12.a Définir un ensemble de colonnes

Un ensemble de colonnes dans une feuille est défini à partir d'une zone de cellules. Cette zone pouvant se réduire à une seule cellule. Les colonnes sont celles qui « traversent » la zone. Reprenant les définitions précédentes, on définit un objet « collection de colonnes » ainsi:

```
Dim MesColonnes As Object, MaColonne  As Object
Dim nbColonnes As Long, nbLignes As Long

UneZone =  MaFeuille.getCellRangeByName("B1:C6")
MesColonnes = UneZone.Columns

UneCellule = MaFeuille.getCellByPosition(3,5) ' cellule D6
MaColonne = UneCellule.Columns
nbColonnes = UneZone.Columns.Count
nbLignes = UneZone.Rows.Count
```

5.12.b Ajouter ou supprimer des colonnes

Les colonnes sont numérotées à partir de zéro.

Ajouter 3 colonnes groupées, dont la première aura le rang 4 dans la collection :

```
MesColonnes.insertByIndex(4,3)
```

Attention, limitation :

La méthode `insertByIndex` refuse d'insérer à la place de la colonne la plus à droite dans une zone de colonne. Pour y arriver il faut donc se positionner sur cette colonne de droite, et insérer en position zero.

Supprimer 3 colonnes groupées, dont la première a le rang 5 dans la collection :

```
MesColonnes.removeByIndex(5,3)
```

5.12.c Optimiser la largeur d'une colonne

Si vous avez dans une colonne des cellules contenant des textes de longueur différentes, il peut être utile d'ajuster la largeur de la colonne à la valeur nécessaire pour le plus long texte.

```
MaColonne.OptimalWidth = TRUE
```

5.12.d Imposer une largeur de colonne

La propriété `Width` définit la largeur de la (ou des) colonne(s), en 1/100 de mm

```
MesColonnes.Width = 3000
```

5.12.e Ajouter ou supprimer des lignes

Les instructions sont quasiment identiques :

```
MesLignes = UneZone.Rows
MaLigne = UneCellule.Rows
rem Ajouter 5 lignes groupées,
rem dont la première aura le rang 2 dans la collection
MesLignes.insertByIndex(2,5)
MesLignes.removeByIndex(5,3)
```

5.12.f Imposer une hauteur de ligne

La propriété `Height` définit la hauteur de la (ou des) ligne(s), en 1/100 de mm

```
MesLignes.Height = 1000
```

5.12.g Cacher des lignes ou des colonnes

On peut cacher à l'utilisateur une colonne ou une ligne. L'exemple suivant cache les lignes et colonnes de la zone B2:C6.

```
Dim MaFeuille As Object, UneZone As Object
Dim MesColonnes As Object, MesLignes As Object

MaFeuille = ThisComponent.Sheets(0)
Unezone =  MaFeuille.getCellRangeByName("B2:C6")
MesColonnes = Unezone.Columns
MesColonnes.IsVisible = false
MesLignes = UneZone.Rows
MesLignes.IsVisible = false
```

Comme pour la plupart des propriétés, `IsVisible` peut être lue pour savoir si une colonne ou une ligne est cachée. Exemple :

```
UneCellule = MaFeuille.getCellByPosition(3,5) ' cellule D6
if UneCellule.Rows.IsVisible and UneCellule.Columns.IsVisible then
  print "La cellule est visible"
else
  print "La cellule est cachée"
end if
```

5.13 Manipuler des données en tableau

Il est assez pénible de lire/écrire successivement dans plusieurs cellules. Or dans bien des cas, on doit effectuer des traitements sur toutes les cellules d'une zone.

Les instructions `getDataArray` et `setDataArray` permettent d'effectuer les traitements dans une variable tableau et de recopier les valeurs depuis/vers la zone de cellules.

L'exemple qui suit ajoute une valeur numérique au contenu de chacune des cellules de la zone sélectionnée par l'utilisateur. On suppose que les cellules contiennent des nombres ou sont vides.

```
Dim MonDocument As Object
Dim UneZone As Object
Dim x1 As Long, y1 As Long
Dim mesValeurs, vLigne, UnElement

UneZone =  ThisComponent.currentSelection
```

```
mesValeurs = UneZone.getDataArray
for y1 = LBound(mesValeurs) to UBound(mesValeurs)
  vligne = mesValeurs(y1)
  for x1 = LBound(vligne) to UBound(vligne)
    UnElement = vligne(x1) +2000 +10*x1 +100*y1
    vligne(x1) = UnElement
  next
  mesValeurs(y1) = vligne
next
UneZone.setDataArray(mesValeurs)
```

Pour la démonstration, le résultat dans chaque cellule reflète la valeur de x1 et y1.

Cet exemple ne fonctionne que si les variables *mesValeurs* et *vLigne* sont de type Variant. Remarquez qu'elles sont déclarées comme des variables simples, alors que l'utilisation les définira comme des tableaux : c'est une possibilité cachée des Variant de OpenOffice Basic.

Il n'est pas possible de manipuler directement un élément dans *mesValeurs*, on doit utiliser des variables intermédiaires. Les fonctions `LBound` et `UBound` sont expliquées dans l'aide en ligne de Basic.

De manière identique, on va modifier une zone de cellules contenant des textes :

```
Dim UneZone As Object
Dim x1 As Long, y1 As Long
Dim mesValeurs, vLigne, UnElement

UneZone = ThisComponent.currentSelection
mesValeurs = UneZone.getDataArray
for y1 = LBound(mesValeurs) to UBound(mesValeurs)
  vligne = mesValeurs(y1)
  for x1 = LBound(vligne) to UBound(vligne)
    UnElement = vligne(x1)& " x1=" & x1 & "  y1=" & y1
    vligne(x1) = UnElement
  next
  mesValeurs(y1) = vligne
next
UneZone.setDataArray(mesValeurs)
```

5.14 Curseur de cellule

Remarque : ce mécanisme semble défectueux, et il est avantageusement remplacé par le positionnement direct sur une cellule.

Avec un curseur de cellule, on peut se déplacer de cellule en cellule.

Méthode	*Signification*
`createCursor`	pour balayer toute la feuille
`createCursorByRange()`	pour balayer une zone de cellules
`gotoStart`	aller à la première cellule non vide
`gotoEnd`	depuis une zone non vide, aller à la dernière cellule non vide
`gotoOffset()`	déplacement par rapport à la position actuelle
`gotoPrev`	aller à la cellule précédente non protégée (celle à gauche en général)
`gotoNext`	aller à la cellule suivante non protégée (celle à droite en général)

Exemple :

```
Dim MonCurseurCellule As Object, UneCellule As Object
```

```
MonCurseurCellule = MaFeuille.createCursor
MonCurseurCellule.gotoStart
MonCurseurCellule.gotoNext
UneCellule = MonCurseurCellule.getCellByPosition(0,0)
UneCellule.String = "Bonjour"

MonCurseurCellule.gotoEnd
rem descendre d'une ligne
MonCurseurCellule.gotoOffset(0, 1)
MonCurseurCellule.getCellByPosition(0,0).String = "Bonsoir"
```

5.15 Appeler des fonctions Calc dans une macro

L'exemple suivant appelle la fonction ADRESSE (voir aide en ligne de Calc, fonctions de la catégorie Classeur). La macro doit appeler la version anglaise de cette fonction, qui a pour nom ADDRESS.

Cette macro est une fonction qui renvoie une adresse de cellule sous forme de chaîne de caractères, à partir des coordonnées utilisées par l'API. Par exemple avec les coordonnées 2,6 :

- celladdr(2,6,4) renvoie "C7"
- celladdr(2,6,0) renvoie "C7"

```
Function celladdr(col As integer, ligne As Integer, absol As Integer) As String
Dim acceder As Object
Dim params As Variant

acceder = CreateUnoService("com.sun.star.sheet.FunctionAccess")
params = Array(ligne+1, col+1, absol, "")
celladdr = acceder.callFunction("Address", params())
End Function
```

Le nom de la fonction est insensible aux majuscules/minuscules, mais doit être le nom anglais. Les paramètres d'appel de la fonction sont enfermés dans la variable tableau *params*, un élément par argument à passer.

La macro *TraduireFormule* vous aidera à traduire vos formules. Dans un classeur Calc, tapez votre formule française en A1 de la feuille 1, exécutez la macro, la traduction anglaise de la formule se trouve maintenant en A3. Cela marche même si votre formule a des erreurs de syntaxe !

```
Sub TraduireFormule
Dim MonDocument As Object, LesFeuilles As Object
Dim MaFeuille As Object
Dim CelluleLocale As Object, CelluleAngl As Object

MonDocument = ThisComponent
LesFeuilles = MonDocument.Sheets
MaFeuille = LesFeuilles(0)
CelluleLocale =  MaFeuille.getCellRangeByName("A1")
CelluleAngl =    MaFeuille.getCellRangeByName("A3")
CelluleAngl.String = CelluleLocale.Formula
End Sub
```

Exemple, montrant l'intérêt de cette macro :

```
=ALEA.ENTRE.BORNES(56;999)
```

est traduite en :

```
=com.sun.star.sheet.addin.Analysis.getRandbetween(56;999)
```

D'autres fonctions de Calc ont pour argument une plage de cellules. En Basic cet argument est un tableau de même structure que celui obtenu avec un `getDataArray` (voir chapitre 5.13).

5.16 Trier une table

La méthode décrite ici est considérée obsolète depuis la version 1.1 d'OpenOffice.org. Mais elle fonctionne encore dans les pré-version 2.0. Le livre Programmation OpenOffice.org décrit la méthode actuellement recommandée.

5.16.a Tri par colonnes

Pour trier une table qui se trouve dans une feuille Calc il est nécessaire de remplir plusieurs descripteurs; pour un tri classique par colonnes, on remplit :

- d'une part un descripteur pour chaque colonne de tri qui précise :
    - de quelle colonne il s'agit,
    - et si le tri est ascendant ou descendant;
    - l'ensemble des ces descripteurs de colonnes constitue un tableau dont
        - l'index 0 indique la colonne à trier en premier,
        - l'index 1 la colonne pour départager les ex-aequo du premier critère tri,
        - l'index 2 la colonne pour départager les ex-aequo restants après le deuxième critère de tri.
- d'autre part un descripteur général de tri, qui précise :
    - si on trie des lignes ou des colonnes,
    - si la table contient une ligne d'en-tête (anglais : `Header`),
    - si le tri est par colonne ou par ligne
    - et l'ensemble des colonnes concernées, obtenu avec le descripteur précédent.

Ensuite, le tri est lancé avec la méthode `Sort`.

Voici un exemple de tri d'une table dont la ligne 1 contient les en-têtes de colonnes, à trier selon les colonnes B en ordre descendant, puis A en ascendant, puis D en ascendant. La zone à trier est une plage nommée, qui contient l'ensemble de la table.

```
Dim MaFeuille As Object, MaZone As Object
Dim ConfigTri(2) As New com.sun.star.util.SortField
Dim DescrTri(2) As New com.sun.star.beans.PropertyValue

MaFeuille = ThisComponent.Sheets.getByName("F_Tris")
MaZone = MaFeuille.getCellRangeByName("ZoneDeTris")

ConfigTri(0).Field = 1 ' colonne B
ConfigTri(0).SortAscending = false
ConfigTri(1).Field = 0 ' colonne A
ConfigTri(1).SortAscending = true
ConfigTri(2).Field = 3 ' colonne D
ConfigTri(2).SortAscending = true

DescrTri(0).Name = "SortFields"
DescrTri(0).Value =  ConfigTri()
DescrTri(1).Name = "Orientation"
DescrTri(1).Value = com.sun.star.table.TableOrientation.ROWS
DescrTri(2).Name = "ContainsHeader"
DescrTri(2).Value = true

MaZone.Sort(DescrTri())
```

Remarques

1. Dans l'instruction `Dim ConfigTri(2)` le numéro d'index maximal correspond au nombre de colonnes servant de critère de tri. Si vous triez selon une seule colonne vous écrirez `Dim ConfigTri(0)`
2. Si une colonne de tri contient des textes, le tri distingue les majuscules des minuscules.
3. La valeur donnée pour l'orientation est bien correcte pour un tri « par colonne » !
4. Les arguments de Name doivent être écrits avec les majuscules/minuscules indiquées.
5. Les parenthèses vides sont indispensables dans la dernière instruction.
6. Comme avec l'interface utilisateur, il n'est pas possible de trier sur plus de 3 critères à la fois.

5.16.b Tri par lignes

Dans le tri de lignes, l'en-tête éventuel est la colonne A, et le tri s'effectue par rapport au contenu d'une, ou plusieurs lignes : inclinez la tête de 90° sur votre gauche, vous retrouvez le tableau classique...

Reprise de l'exemple précédent, en configuration de tri de lignes :

```
Dim MaFeuille As Object, MaZone As Object
Dim ConfigTri(2) As New com.sun.star.util.SortField
Dim DescrTri(2) As New com.sun.star.beans.PropertyValue

MaFeuille = ThisComponent.Sheets.getByName("F_Tris")
MaZone = MaFeuille.getCellRangeByName("ZoneDeTris")

ConfigTri(0).Field = 1 ' ligne 2
ConfigTri(0).SortAscending = false
ConfigTri(1).Field = 0 ' ligne 1
ConfigTri(1).SortAscending = true
ConfigTri(2).Field = 3 ' ligne 4
ConfigTri(2).SortAscending = true

DescrTri(0).Name = "SortFields"
DescrTri(0).Value =  ConfigTri()
DescrTri(1).Name = "Orientation"
DescrTri(1).Value =  com.sun.star.table.TableOrientation.COLUMNS
DescrTri(2).Name = "ContainsHeader"
DescrTri(2).Value = true

MaZone.Sort(DescrTri())
```

5.17 Les dessins dans Calc

Ici, on appelle une forme (anglais : shape) un dessin élémentaire, par exemple un rectangle, une ellipse, etc.

5.17.a Insérer une forme dessinée

Une forme est insérée dans une page de dessin. Il existe autant de pages de dessin que de feuilles dans le tableur Calc, avec la même numérotation à partir de zéro. Cet exemple minimal insère un rectangle dans la troisième feuille du tableur :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim PageDessin As Object
Dim MaForme As Object
Dim Taille1 As New com.sun.star.awt.Size

Taille1.Width = 5400
Taille1.Height = 2530
```

```
MonDocument = ThisComponent

PageDessin = MonDocument.DrawPages(2) ' page de la feuille Calc de rang 2
MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
MaForme.Size = Taille1
PageDessin.add(MaForme)
```

L'instruction `Dim` *Taille1* crée un objet de type Taille (en anglais : Size) qui servira à dimensionner la future forme. Basic ne permet pas de modifier directement la propriété `Size` d'une forme.

L'argument de `createInstance` est une horrible constante qui précise le type de forme dessinée.

Les principales formes élémentaires disponibles dans `com.sun.star.Drawing` sont :

Forme	*Signification*
`CaptionShape`	Etiquette
`ClosedBezierShape`	Courbe de Bézier fermée
`ConnectorShape`	Connecteur
`EllipseShape`	Ellipse ou cercle
`GraphicObjectShape`	Objet graphique
`LineShape`	Ligne
`MeasureShape`	Ligne de cote
`OpenBezierShape`	Courbe de Bézier ouverte
`PolyLineShape`	Ligne brisée
`PolyPolygonShape`	Polygone
`RectangleShape`	Rectangle ou carré
`TextShape`	Texte

Une fois le dessin ajouté à la page, on peut changer ses propriétés, listées plus loin.

5.17.b Ancrage de la forme

La forme obtenue avec les instructions de l'exemple précédent est positionné dans le coin en haut à gauche de la feuille, ancré à la cellule A1.

L'interface utilisateur permet (clic droit sur le dessin) de choisir un ancrage à la page ou à la cellule. Avec un ancrage à la cellule, on peut déplacer à la souris la forme vers une autre cellule.

Le positionnement par macro utilise en fait un ancrage intermédiaire entre « page » et « cellule ».

5.17.c Positionner la forme

Le positionnement fait appel à une variable intermédiaire, qui définit la position d'un point, en coordonnées x et y. Ce sera la position du coin haut-gauche de la forme. Les coordonnées sont exprimées en 1/100 de mm, relativement au coin haut-gauche de la feuille.

```
Dim MonDocument As Object, LesFeuilles As Object
Dim PageDessin As Object
Dim MaForme As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Posit1 As New com.sun.star.awt.Point

Taille1.Width = 5400
Taille1.Height = 2530
Posit1.x = 5700 ' 55 mm vers la droite
```

```
Posit1.y = 8350 ' 83,5 mm vers le bas

MonDocument = ThisComponent

PageDessin = MonDocument.DrawPages(2) ' page de la feuille Calc de rang 2
MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
MaForme.Size = Taille1
MaForme.Position = Posit1
PageDessin.add(MaForme)
```

Le résultat de ces instructions est une forme ancrée à une cellule de la feuille, par exemple B18. Que s'est-il passé ? 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.

En pratique, ceci veut dire 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 de la colonne ou la hauteur de la ligne contenant la cellule d'ancrage, les dimensions de la forme changeront.
Si on modifie la largeur d'une colonne précédente ou la hauteur d'une ligne précédente, la position de la forme changera.

5.17.d Autres fonctionnalités autour de la forme

Les autres fonctionnalités sont identiques pour Calc et les autres applications. Elles sont décrites dans un chapitre commun :

- Propriétés communes
- Dessin de différentes formes
- Ecriture de texte dans la forme
- Donner un nom à une forme

5.18 Les images dans Calc

5.18.a Insérer une image

Note : dans l'API version anglaise, le terme Graphic est utilisé pour désigner une image « bitmap » ou vectorielle. Il n'y a aucun rapport avec le concept français de graphique.

Cet exemple insère une image :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim PageDessin As Object
Dim MonImage As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent

PageDessin = MonDocument.DrawPages(2) ' page de la troisième feuille Calc
MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")

PageDessin.add(MonImage)

Taille1.Width  = 6000 ' largeur en 1/100 de mm
Taille1.Height = 6000 ' largeur en 1/100 de mm
MonImage.Size = Taille1
```

Si l'image originale n'a pas les mêmes proportions, l'image sur le document sera déformée. Nous verrons plus loin comment redimensionner en gardant les proportions originales.

Ces instructions n'insèrent pas l'image dans le document Calc, mais seulement un lien vers l'image. Si vous déplacez, renommez, supprimez le fichier image référencé, le document affichera une case de lien brisé avec l'adresse URL du fichier.

Ceci est une limitation actuelle de l'API.

Pour effectivement insérer dans le document des images insérées par macro, suivre cette procédure :

1. Menu Edition > Liens...
   - sélectionner l'ensemble des liens images
   - cliquer sur le bouton Déconnecter; confirmer
2. Sauver le document.

5.18.b Positionner une image

L'ancrage et le positionnement d'une image et d'une forme sont identiques. Reportez-vous au chapitre concernant le positionnement des formes dans Calc.

5.18.c Dimensionner une image

Pour dimensionner l'image sans changer les proportions, on récupère les dimensions de l'image en pixels. Ceci n'est possible qu'après l'instruction `PageDessin.add(MonImage)`. La proportion calculée sert à fixer la hauteur en fonction de la largeur. Ces deux dimensions sont en 1/100 de mm.

Cet exemple place l'image dans la feuille et redimensionne l'image à une largeur choisie, tout en gardant ses proportions :

```
Dim MonDocument As Object, LesFeuilles As Object
Dim PageDessin As Object
Dim MonImage As Object
Dim LeBitmap As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Posit1 As New com.sun.star.awt.Point
Dim Proportion As Double

MonDocument = ThisComponent

PageDessin = MonDocument.DrawPages(2) ' page de la feuille Calc de rang 2
MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")

Posit1.x = 3000
Posit1.y = 6000
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonImage.Position = Posit1

PageDessin.add(MonImage)

LeBitmap = MonImage.GraphicObjectFillBitmap
Taille1 = LeBitMap.Size ' taille en pixels !
Proportion = Taille1.Height / Taille1.Width
Taille1.Width = 6000 ' largeur en 1/100 de mm
Taille1.Height = Taille1.Width * Proportion
MonImage.Size = Taille1
```

5.18.d Insérer plusieurs images

A chaque insertion d'image il est nécessaire d'obtenir un nouveau GraphicObjectShape, même si on insère plusieurs fois la même image, comme dans l'exemple qui suit. Il comporte un sous-programme facilitant le redimensionnement.

```
Sub Inserer2ImagesIdentiques
Dim MonDocument As Object, LesFeuilles As Object
Dim PageDessin As Object
Dim MonImage As Object
Dim Posit1 As New com.sun.star.awt.Point

MonDocument = ThisComponent

PageDessin = MonDocument.DrawPages(2) ' page de la feuille Calc de rang 2
MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
Posit1.x = 3000
Posit1.y = 6000
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonImage.Position = Posit1
PageDessin.add(MonImage)
LargeProp(MonImage, 6000)

MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
Posit1.x = 10000
Posit1.y = 8000
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonImage.Position = Posit1
PageDessin.add(MonImage)
LargeProp(MonImage, 3500)

End Sub


Sub LargeProp( UneImage As Object, Largeur As Long)
Dim LeBitmap As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Proportion As Double

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
```

5.18.e Trouver une image par son nom

Pour pouvoir retrouver une image, un bon moyen est de la nommer. Une fois l'image insérée dans le document, il suffit d'utiliser la propriété `Name` :

```
MonImage.Name = "Mon chien"
```

Toutes les images (et les formes) d'une page sont accessibles par un index, exemple :

```
MonImage = PageDessin(3)
```

Ceci est peu pratique, mais dans Calc il n'existe pas de méthode API pour retrouver une forme par son nom.

Pas de problème, nous allons utiliser la fonction *FindObjectByName* décrite dans un autre chapitre. Avec cette fonction, nous pouvons retrouver l'image et la modifier :

```
Dim MonDocument As Object
Dim MonImage As Object
Dim PageDessin As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent
PageDessin = MonDocument.DrawPages(2)
MonImage = FindObjectByName( PageDessin, "Mon chien")
if IsNull(MonImage) then
  print "Nom inexistant !"
else
```

```
  Taille1.Width = 6000 ' largeur en 1/100 de mm
  Taille1.Height = 3000 ' hauteur en 1/100 de mm
  MonImage.Size = Taille1
end if
```

Attention : cette méthode retrouve un objet quelconque sur la page, aussi évitez de donner le même nom à une forme et à une image de la même page.

6 Traitement de texte Writer

6.1 Trouver le texte

Pour travailler sur un texte de Writer nous avons besoin de le désigner. Le fait de désigner le document lui-même est insuffisant. En effet, un document Writer contient différents objets, dont le texte ordinaire, les en-tête et pied de page, ou des cadres contenant du texte.

Pour une macro qui travaille sur le texte ordinaire du document en cours, on écrit :

```
Dim MonDocument As Object, MonTexte As Object
MonDocument = ThisComponent
MonTexte = MonDocument.Text
```

6.2 Gérer les curseurs

6.2.a Curseur visible, curseur d'écriture

Il faut bien comprendre qu'il existe deux types de curseurs :

1. le curseur visible, c'est celui que vous voyez avancer quand vous tapez du texte dans Writer; il sera décrit plus loin
2. le curseur d'écriture, qui est utilisé par la macro pour désigner le texte qu'elle manipule.

Pour insérer du texte ordinaire à partir du début du document Writer, on définit ainsi le curseur d'écriture :

```
MonCurseur= MonTexte.createTextCursor
```

Par contre, pour insérer du texte ailleurs dans un texte existant nous avons besoin de déplacer ce curseur.

6.2.b Déplacer le curseur d'écriture

Voici deux exemples de déplacement du curseur d'écriture :

```
MonCurseur.goRight(1, false) ' se déplacer à droite d'un caractère
MonCurseur.goLeft(2, true) ' sélectionner les 2 caractères précédents
```

Faites bien attention : le curseur d'écriture se déplace, mais le curseur visible, lui, n'est pas déplacé.

Le deuxième argument de `goRight` et `goLeft` signifie :

- false = déplacer le curseur
- true = déplacer le curseur en étendant la sélection (comme en glissant avec la souris). La sélection de travail est invisible.

Liste des instructions pour le curseur

Les instructions pour déplacer le curseur ont une syntaxe identique:

Méthode	*Signification*
`goRight(n, false)`	déplacer de n caractères à droite
`goLeft(n, false)`	déplacer de n caractères à gauche
`gotoStart(false)`	déplacer au début du texte complet
`gotoEnd(false)`	déplacer à la fin du texte complet
`gotoStartOfParagraph(false)`	déplacer au début du paragraphe en cours

Méthode	*Signification*
`gotoEndOfParagraph(false)`	déplacer à la fin du paragraphe en cours
`gotoNextParagraph(false)`	déplacer au début du paragraphe suivant
`gotoPreviousParagraph(false)`	déplacer au début du paragraphe précédent
`gotoNextWord(false)`	déplacer au début du mot suivant
`gotoPreviousWord(false)`	déplacer au début du mot précédent
`gotoStartOfWord(false)`	déplacer au début du mot courant
`gotoEndOfWord(false)`	déplacer à la fin du mot courant
`gotoNextSentence(false)`	déplacer au début de la phrase suivante
`gotoPreviousSentence(false)`	déplacer au début de la phrase précédente
`gotoStartOfSentence(false)`	déplacer au début de la phrase courante
`gotoEndOfSentence(false)`	déplacer à la fin de la phrase courante

Instructions pour manipuler les extrémités du curseur

Méthode	*Signification*
`collapseToStart`	amène la fin du curseur sur la position du début de curseur
`collapseToEnd`	amène le début du curseur sur la position de fin du curseur

Toutes ces instructions sont en réalité des fonctions, dont le résultat est vrai si l'action a pu être exécutée, ou faux sinon. On n'est pas obligé d'utiliser ce résultat.

Il existe des fonctions pour déterminer la position du curseur :

Fonction	*Signification*
`isStartOfParagraph`	résultat vrai si le curseur est au début d'un paragraphe
`isEndOfParagraph`	résultat vrai si le curseur est à la fin d'un paragraphe
`isStartOfWord`	résultat vrai si le curseur est au début du mot courant
`isEndOfWord`	résultat vrai si le curseur est à la fin du mot courant
`isStartSentence`	résultat vrai si le curseur est en début de phrase
`isEndOfSentence`	résultat vrai si le curseur est en fin de phrase
`isCollapsed`	résultat vrai si le début et la fin du curseur sont identiques

Les méthodes relatives aux mots et phrases peuvent donner des résultats surprenants. Voyez éventuellement plus loin les remarques concernant la notion de mot.

6.2.c 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.

Obtenir le curseur visible

```
Dim MonDocument As Object
Dim CurseurVisible  As Object
```

```
MonDocument = ThisComponent
CurseurVisible = MonDocument.CurrentController.ViewCursor
```

Notez que *CurseurVisible* a certaines des possibilités d'un curseur d'écriture comme *MonCurseur*, mais si on déplace la position de *CurseurVisible*, ce ne sera pas reflété à l'écran. Pour utiliser toutes les possibilités, le mieux est d'initialiser un curseur d'écriture à la position du curseur visible.

Curseur d'écriture et curseur visible

Nous allons positionner un curseur d'écriture à la position du curseur visible. Mais le curseur visible peut sélectionner un point, ou une zone qui a un début et une fin.
Positionnons le curseur d'écriture au début de la zone du curseur visible :

```
Dim MonDocument As Object, MonTexte As Object
Dim CurseurVisible As Object, MonCurseur  As Object

MonDocument = ThisComponent
CurseurVisible = MonDocument.currentcontroller.ViewCursor
MonTexte = MonDocument.Text
MonCurseur = MonTexte.createTextCursorByRange(CurseurVisible.Start)
```

Positionnons le curseur d'écriture à la fin de la zone du curseur visible :

```
MonCurseur = MonTexte.createTextCursorByRange(CurseurVisible.End)
```

6.2.d Zone sélectionnée par l'utilisateur

Il n'est pas utile d'initialiser un curseur d'écriture pour certaines tâches simples. Voici comment récupérer le texte sélectionné par l'utilisateur et le remplacer :

```
Dim MonDocument As Object
Dim CurseurVisible As Object
Dim TexteDeLaSelection As String

MonDocument = ThisComponent
CurseurVisible = MonDocument.currentcontroller.ViewCursor
TexteDeLaSelection = CurseurVisible.String
CurseurVisible.String = " nouveau texte "
```

Dans la séquence suivante on met en gras la zone de texte sélectionnée par l'utilisateur (la mise en gras est expliquée plus loin) :

```
Dim MonDocument As Object
Dim CurseurVisible As Object

MonDocument = ThisComponent
CurseurVisible = MonDocument.currentcontroller.ViewCursor
CurseurVisible.CharWeight = com.sun.star.awt.FontWeight.BOLD
```

Notez que la séquence ci-dessus est aussi valide si plusieurs zones non contiguës sont sélectionnées par l'utilisateur : toutes les zones sont mises en gras !

Si l'utilisateur sélectionne plusieurs zones à la fois, il est plus souvent utile de traiter successivement chaque zone. La propriété `CurrentSelection` 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`.

```
Dim MonDocument As Object
Dim LesSelections As Object
Dim selx As Integer, UneZone As Object

MonDocument = ThisComponent
LesSelections = MonDocument.CurrentSelection
for selx = 0 to LesSelections.Count -1
  UneZone = LesSelections(selx)
  rem mettre en gras chaque zone sélectionnée
```

```
  UneZone.CharWeight = com.sun.star.awt.FontWeight.BOLD
next
```

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` n'est pas exécutée.

Attention, la variable *UneZone* n'est pas un curseur. Pour obtenir un curseur à partir d'une zone sélectionnée, faire :

```
Dim UnCurseur As Object
UnCurseur = MonTexte.createTextCursorByRange(UneZone)
```

6.2.e Sélectionner visiblement une zone

Le but est de sélectionner une zone par macro, et d'afficher cette sélection à l'utilisateur. Ce n'est pas en modifiant *CurseurVisible* que vous y arriverez. Il faut une instruction spéciale, qui utilise la position du curseur mis en argument :

```
MonCurseur.gotoNextWord(false)
MonCurseur.gotoEndOfWord(true) ' selection d'un mot
' Transférer la sélection sur le curseur visible
MonDocument.CurrentController.Select(MonCurseur)
```

Si vous avez exécuté cette macro dans l'EDI, mettez en avant-plan la fenêtre Writer pour voir la sélection du mot.

6.3 Ecrire du texte

6.3.a Lire, écrire

On peut donc sélectionner avec le curseur une zone de plusieurs caractères; ceci fait, on récupère le texte sélectionné ainsi :

```
Dim tx3 As String
tx3 = MonCurseur.String
```

Inversement, on peut écraser un texte sélectionné avec un nouveau texte :

```
MonCurseur.String = "Bonsoir"
```

Attention : si vous utilisez ce principe avec l'objet Texte, vous écrasez tout texte existant :

```
MonTexte.String = "tout est écrasé par ceci"
```

Notez que Basic limite les chaînes de caractères à 64 000 caractères. Ces méthodes ne marchent donc pas pour des textes plus longs.

Habituellement nous écrirons du texte ainsi :

```
MonTexte.insertString( MonCurseur, "Ceci est un texte", false)
```

Après cette instruction, le curseur d'écriture est positionné à la fin du texte qu'elle a écrit.

Dans la procédure `insertString` le troisième argument signifie :

- `false` = insérer dans le texte
- `true` = écraser dans le texte

6.3.b Caractères spéciaux dans un texte

Insérer une fin de paragraphe de texte :

```
Dim FinParagraphe As Integer
FinParagraphe = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
```

```
MonTexte.insertControlCharacter(MonCurseur, FinParagraphe, false)
```

Le dernier argument (false) a la même signification que pour `insertString` .

Nous avons employé la variable intermédiaire *FinParagraphe* pour simplifier le codage.

Il existe une liste d'horribles constantes similaires :

Constante	*Signification*
`PARAGRAPH_BREAK`	fin de paragraphe
`LINE_BREAK`	retour à la ligne
`HARD_HYPHEN`	tiret insécable
`SOFT_HYPHEN`	tiret conditionnel
`HARD_SPACE`	espace insécable
`APPEND_PARAGRAPH`	insérer un paragraphe après celui en cours et se positionner au début

Supprimer des paragraphes

Cet exemple utilise plusieurs des notions décrites jusqu'ici. Il parcourt un texte complet et supprime les marques de paragraphe en remplaçant chacune par un espace.

```
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
```

6.3.c 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 :

```
Dim genreSautpage As Integer
genreSautpage= com.sun.star.style.BreakType.PAGE_AFTER
MonCurseur.breakType= genreSautpage
```

Vous avez reconnu une horrible constante. Différents types de saut de page sont possibles :

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 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.

6.3.d Exemples de manipulation de texte

Cette macro va corriger vos fautes de frappe; elle inverse les deux caractères de part et d'autre de la position pointée par l'utilisateur :

```
Sub Inverser2car

Dim MonDocument As Object, MonTexte As Object
Dim CurseurVisible As Object, MonCurseur  As Object
Dim tx0, tx1, tx2 As String

MonDocument = ThisComponent
MonTexte = MonDocument.Text
rem on suppose le curseur visible positionné entre les deux caractères
CurseurVisible = MonDocument.currentcontroller.ViewCursor
MonCurseur = MonTexte.createTextCursorByRange(CurseurVisible.Start)

MonCurseur.goRight(1, false) ' position après le 2eme caractère
MonCurseur.goLeft(2, true) ' sélectionner les 2 caractères précédents
tx0 = MonCurseur.String
tx1 = Left(tx0, 1)
tx2 = Right(tx0, 1)
tx0 = tx2 & tx1  ' inverser les 2 caractères
MonCurseur.String = tx0
'restituer la position initiale du curseur visible
CurseurVisible.GoLeft(1,false)

End Sub
```

Reprenons l'exemple de macro pour supprimer des paragraphes. Cette nouvelle version ne supprime les paragraphes que dans la zone sélectionnée par l'utilisateur :

```
Dim MonDocument As Object, MonTexte As Object
Dim CurseurVisible As Object, MonCurseur  As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
CurseurVisible = MonDocument.currentcontroller.ViewCursor
rem creer un curseur de travail à la position 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
```

La fonction `compareRegionEnds` compare les positions de fin dans les deux curseurs.

Remarque : si l'utilisateur sélectionne plusieurs zones, la macro ne supprime rien.

6.3.e Astuce : utiliser les signets pour écrire un texte

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.

Vous pouvez vous simplifier considérablement l'écriture de texte par macro si vous remplissez un document pré-défini (ou un modèle de document) qui contiendra la structure fixe du document, et des signets pour repérer les endroits que la macro doit remplir. Evidemment 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 le signet nommé «ecrire_ici», puis écrit « Texte nouveau « :

```
MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur = MonDocument.Bookmarks.getByName("ecrire_ici").getAnchor
MonTexte.insertString( MonCurseur, " Texte nouveau ", false)
```

Le nom du signet doit être écrit exactement comme dans sa définition. L'instruction 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
```

6.4 Appliquer un style ou un formatage

6.4.a Appliquer un style à un paragraphe

Le moyen le plus efficace de formater un texte avec une macro est de lui appliquer des styles que vous avez définis au préalable. Si vous avez besoin de créer un nouveau document dans votre projet de macro, vous pouvez définir manuellement un document modèle avec les styles dont vous avez besoin, puis utiliser ce modèle pour créer le nouveau document.

La macro suivante fait exactement comme si vous affectiez vous-même le style « Titre 4 » au deuxième paragraphe du document en cours. Testez-la avec un petit document de trois paragraphes.

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor
MonCurseur.gotoNextParagraph(false)
rem ici on est au deuxieme paragraphe du texte existant

MonCurseur.paraStyleName= "Titre 4"
```

Le curseur d'écriture peut être dans une position quelconque dans le paragraphe. J'aurais pu le décaler de 5 caractères, par exemple.

On peut aussi connaître le style d'un paragraphe :

```
print MonCurseur.paraStyleName
```

Attention

Si vous récupérez le nom du style en cours, pour les styles standards fournis avec OOo vous obtiendrez le nom anglais, même avec une version localisée de OOo. Dans l'exemple ci-dessus, on affecte le style « Titre 4 » et on relira le style « Heading 4 ». Par contre les styles que vous créez n'ont évidemment qu'un seul nom. Donc, en cas de doute, lisez le nom du style avec une petite macro.

Deuxième problème potentiel : la chaîne de caractère du style que vous affectez à un paragraphe doit reproduire exactement le nom du style existant : majuscules, minuscules, accents. Sinon le style du paragraphe restera inchangé.

6.4.b Affecter un style à un caractère

Pour changer le style de certains caractères, on les sélectionne et on applique `CharStyleName` :

```
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"
```

6.4.c Affecter un formatage à un caractère

Ceci est l'équivalent d'appliquer Gras, Italique, Souligné, etc, à un ou plusieurs caractères.

Exemple sur le deuxième mot du deuxième paragraphe :

```
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
```

Les propriétés de caractère sont listées dans l'API : `CharacterProperties`. Les principales :

Propriété	***Signification***
`CharStyleName`	Chaîne de caractère, nom du style de caractère (respecter majuscules et minuscules)
`CharFontName`	Chaîne de caractères, nom de la police (respecter majuscules et minuscules)
`CharHeight`	La taille du caractère, en points
`CharWeight`	La « grasse » du caractère
`CharShadowed`	booléen; valeur true pour mettre une ombre au caractère
`CharPosture`	Caractère penché ou droit
`CharUnderline`	Pour souligner le caractère
`CharColor`	Couleur du caractère
`CharBackColor`	Couleur de fond du caractère
`CharCrossedOut`	booléen; valeur true pour barrer le caractère
`CharEscapement`	Pour un caractère en exposant ou en indice
`CharEscapementHeight`	Pour un caractère en exposant ou en indice

Certaines propriétés utilisent des horribles constantes comme pour `CharWeight` dans l'exemple ci-dessus. Voici le détail des principales propriétés :

- `CharWeight` constantes comme : `com.sun.star.awt.FontWeight.XXXX`

Constante	Signification
NORMAL	Normal
BOLD	Gras **exemple**
THIN	Maigre (selon disponibilité dans la police)
ULTRABOLD	Très gras (selon disponibilité dans la police)

D'autres valeurs peu courantes existent, voir l' API.

- CharPosture constantes comme : com.sun.star.awt.FontSlant.XXXX

Constante	Signification
NONE	caractère droit
ITALIC	caractère en italique *exemple*

- CharUnderline constantes comme : com.sun.star.awt.FontUnderline.XXXX

Constante	Signification
NONE	normal, non souligné
SINGLE	souligné simple exemple
DOUBLE	souligné double exemple
DOTTED	souligné pointillé exemple
WAVE	souligné ondulé exemple
DASH	souligné tireté exemple
DASHDOT	souligné tiret-point exemple
BOLD	souligné gras exemple

D'autres valeurs peu courantes existent, voir l' API.

Caractère en exposant ou en indice

Mettre un caractère en exposant ou en indice nécessite deux propriétés de caractère : CharEscapement et CharEscapementHeight.

- CharEscapement spécifie la hauteur du caractère par rapport à un caractère normal, en pourcentage; une valeur positive pour un exposant, une valeur négative pour un indice.
- CharEscapementHeight spécifie la taille relative du caractère, en pourcentage, par rapport à la taille du caractère normal.

Exemple :

```
MonCurseur.gotoEndOfWord(false)
MonCurseur.goLeft(1, true) ' sélectionner un caractère
MonCurseur.CharEscapement = 20 ' hauteur du caractère / ligne
MonCurseur.CharEscapementHeight = 70 ' prop hauteur caractère / taille normale
```

Vous pouvez vérifier le résultat avec l'interface utilisateur : sélectionner le caractère modifié par la macro, menu Format > Caractères... onglet Position.

Remarque : la documentation de l'API inverse les descriptions de ces deux propriétés.

6.4.d Remettre le formatage du style du paragraphe

Pour tout nettoyer, c'est-à-dire ne laisser que le formatage propre au style du paragraphe en cours, on applique à une sélection de texte la méthode `setAllPropertiesToDefault`.

```
MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(true) ' selectionner tout un paragraphe
MonCurseur.setAllPropertiesToDefault
```

6.5 Exercice d'écriture de texte

Pour aller plus loin, on va faire un travail pratique. Ouvrez un nouveau document Writer, tapez quelques paragraphes en style « standard ». Puis avec le menu Outils/Macro créez la macro suivante :

```
Option Explicit

Sub Main
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object
Dim FinParagraphe As Integer

MonDocument = ThisComponent
MonTexte = MonDocument.Text
FinParagraphe = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
MonCurseur= MonTexte.createTextCursor

MonTexte.insertString( MonCurseur, "Début de texte", false)
MonTexte.insertControlCharacter( MonCurseur, FinParagraphe , false)
MonTexte.insertString( MonCurseur, "Deuxième ligne", false)
MonTexte.insertControlCharacter( MonCurseur, FinParagraphe , false)
MonTexte.insertString( MonCurseur, "Début paragr 3 ", false)
MonCurseur.paraStyleName= "Titre 4"
MonCurseur.CharColor = 225
MonCurseur.CharShadowed = true
MonTexte.insertString( MonCurseur, " ajout en couleurs", false)

End Sub
```

Laissez le curseur visible à la fin des quelques lignes que vous avez écrites sur le document. Ensuite exécutez la macro en « étape par étape », tout en visualisant le contenu du texte dans le document.

Après « Début de texte » vous constaterez que cette séquence est écrite en tout début de document : c'est dû à l'initialisation faite par `createTextCursor`.

La méthode `paraStyleName` affecte le style « Titre 4 » au paragraphe en cours et à tout le texte actuel du paragraphe.

Les instructions `CharColor` et `CharShadowed` affectent une couleur bleue et une ombre au curseur, c'est-à-dire aux caractères qui seront écrits à partir d'ici – comme si vous le faisiez manuellement -

La dernière instruction exécutée, on voit que seul ce dernier texte est en couleurs.

Variantes

Ré-exécutez la macro, en effectuant des variantes. Pour chaque essai, effacez manuellement le texte écrit précédemment par la macro (cliquez sur le bouton « Annuler » dans la fenêtre du document Write).

- Changez manuellement le style de la première ligne du texte en « corps de texte ». Exécutez la macro : les deux premières lignes garderont ce style, mais la troisième sera en Titre 4.
- Changez ainsi les dernières lignes de la macro :

```
MonCurseur.paraStyleName= "Titre 4"
MonCurseur.gotoStartOfParagraph(false)
MonCurseur.CharColor = 225
MonCurseur.CharShadowed = true
MonTexte.insertString( MonCurseur, " ajout en couleurs", false)
MonCurseur.gotoEndOfParagraph(false)
MonTexte.insertString( MonCurseur, " en noir ", false)
```

L'ajout en couleurs est maintenant en début de paragraphe, le texte ajouté « en noir » suit bien le style en cours à cette position.

- Remplacez `gotoStartOfParagraph` par `gotoStart` et voyez la différence.
- Remplacez aussi `gotoEndOfParagraph` par `gotoEnd` et voyez la différence.
- Remplacez aussi `false` par `true` dans `gotoEnd` et dans le dernier `insertString` ; exécutez pas à pas : à la fin, tout le texte après « ajout en couleurs » est remplacé par le dernier insert, qui s'affiche avec le style de caractère en cours ici.
- Remplacez maintenant le dernier `insertString` par

```
MonCurseur.String = " en noir "
```

Le résultat est identique.

6.6 Tabuler un texte

Insérer une tabulation dans un texte consiste à ajouter le caractère dont la valeur décimale 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, deux solutions :

1. la plus simple et pratique est d'utiliser un style de paragraphe que vous avez défini dans votre document;
2. ou bien, définir laborieusement par programmation les taquets du paragraphe en cours, voir la macro ci-après.

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object, Texte1 As String
Dim NouvParagr As Integer
Dim Tabul As String
Dim PositionTaquet As New com.sun.star.style.TabStop
Dim ListeTaquets(2) As Object

NouvParagr = com.sun.star.text.ControlCharacter.APPEND_PARAGRAPH
Tabul = chr(9)

With PositionTaquet
  .Position = 2500 ' 25 mm ( 2,5 cm )
  .Alignment = com.sun.star.style.TabAlign.LEFT
  .DecimalChar = Asc(",")
  .FillChar = Asc(" ")
  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
```

```
rem Insérer un paragraphe après celui en cours (ici, le premier)
MonTexte.insertControlCharacter(MonCurseur, NouvParagr, false)
MonCurseur.ParaTabStops = ListeTaquets() ' imposer les taquets
Texte1 = "Début"+Tabul+"Tab0"+Tabul+"Tab1"+Tabul+"Tab2"
MonTexte.insertString( MonCurseur, Texte1, false)
```

Vous devez affecter tous les taquets en une seule fois, avec un tableau comportant le nombre de taquets nécessaires. Chaque taquet est constitué de plusieurs valeurs :

Composante	***Signification***
`Position`	La position du taquet par rapport à la marge gauche, en 1/100 de mm.
`Alignment`	Le type de tabulation, sous forme d'une horrible constante décrite plus bas.
`DecimalChar`	Le caractère séparateur décimal (en France, la virgule). Vous devez remplir ce champ avec un entier correspondant à la valeur Unicode du caractère.
`FillChar`	Le caractère de suite, qui remplit les blancs, par exemple un pointillé. Même remarque que pour `DecimalChar`.

Les types de tabulation possibles sont :

Constante	***Signification***
`LEFT`	Le taquet de tabulation se trouvera à gauche du texte à tabuler.
`CENTER`	Le taquet de tabulation se trouvera au centre du texte à tabuler.
`RIGHT`	Le taquet de tabulation se trouvera à droite du texte à tabuler.
`DECIMAL`	Le taquet de tabulation se trouvera sur le séparateur de décimales.
`DEFAULT`	Utiliser le type de tabulation par défaut.

Remarque : les explications de l'API (version du 04/02/2003) sur les types de tabulation sont complètement fausses.

6.7 Recherche et remplacement de texte

6.7.a Remplacer une chaîne partout dans un texte

Ecrivez dans un document Writer plusieurs fois la chaîne de caractères « Annette ». La macro suivante remplace tous les mots « Annette » par « Josette » dans tout le document, sans modifier des chaînes de caractères comme « Jannette ».

```
Dim MonDocument As Object
Dim JeCherche As Object
Dim Nombredefois As Long

MonDocument = ThisComponent
JeCherche= MonDocument.createReplaceDescriptor
with JeCherche
 .SearchString = "Annette"
 .ReplaceString = "Josette"
 .SearchWords = true ' defaut : false
 .SearchCaseSensitive = true 'defaut : false
end with
Nombredefois = MonDocument.replaceAll(JeCherche)
print Nombredefois, "remplacement(s)"
```

Explications :

On crée un objet *JeCherche* qui contiendra tous les paramètres nécessaires à ce remplacement. Son initialisation requiert plusieurs instructions; Pour éviter la répétition lassante de *JeCherche* à

chaque instruction, on utilise la séquence With JeCherche ... end With (littéralement : Avec ...).

Les propriétés du descripteur de remplacement permettent de préciser la recherche :

Propriété	***Signification***
`SearchString`	La chaîne de caractères à rechercher
`ReplaceString`	La chaîne de caractères en remplacement. Par défaut, cette chaîne est vide, ce qui revient à supprimer.
`SearchWords`	booléen; true pour ne rechercher que des mots; par défaut on recherche la séquence de caractère n'importe où.
`SearchCaseSensitive`	booléen; true pour distinguer les majuscules des minuscules dans la recherche; par défaut il n'y a pas de distinction. (1)
`SearchBackwards`	booléen; true pour faire une recherche à reculons (ça peut servir); par défaut on recherche dans le sens normal de lecture.
`SearchRegularExpression`	booléen; true pour faire une recherche avec la méthode des expressions régulières (si ça ne vous dit rien, ne l'utilisez pas); par défaut on recherche une simple égalité de chaîne. (2)
`SearchStyles`	booléen; true pour rechercher des paragraphes d'un style donné par SearchString; par défaut on cherche du texte.
`SearchSimilarity`	booléen; true pour rechercher un texte similaire au texte cherché (2) (3)
`SearchSimilarityAdd`	Nombre de caractères à ajouter pour retrouver le texte cherché
`SearchSimilarityRemove`	Nombre de caractères à retrancher pour retrouver le texte cherché
`SearchSimilarityExchange`	Nombre de caractères à changer pour retrouver le texte cherché
`SearchSimilarityRelax`	booléen; true pour essayer toute combinaison des trois critères précédents qui permette de retrouver le texte cherché

Notes :

(1) La recherche sans distinction majuscule/minuscule distingue cependant les caractères accentués.

(2) Fonctionne seulement à partir de la version 1.1 de OpenOffice

(3) Remplir tous les critères associés, les valeurs par défaut ne sont pas zéro.

6.7.b Remplacer un nom de style partout dans un texte

Cette macro remplace tous les utilisations du style de paragraphe « Titre 4 » par le style «Titre 5» :

```
Dim MonDocument As Object
Dim JeCherche As Object
Dim MonCurseur  As Object
Dim unstyle as string

MonDocument = ThisComponent
JeCherche= MonDocument.createReplaceDescriptor
with JeCherche
 .SearchString = "Titre 4"
 .ReplaceString = "Titre 5"
 .SearchStyles = true
end with
```

```
MonDocument.replaceAll(JeCherche)
```

Relisez à ce sujet les remarques sur les noms de styles de paragraphe.

6.7.c Chercher et modifier du texte

Les méthodes de ce chapitre permettent d'effectuer des modifications quelconques sur les élements cibles, par exemple changer le formatage de certains mots. De plus, la recherche peut être limitée à une partie du texte entier.

Chercher dans tout le texte

Cette macro recherche les mots « hello » dans tout le texte et les met en gras.

```
Dim MonDocument As Object
Dim JeCherche As Object
Dim posTrouve As Object

MonDocument = ThisComponent
JeCherche= MonDocument.createSearchDescriptor

with JeCherche
 .SearchString = "hello"
 .SearchWords = true
end with

posTrouve = MonDocument.findFirst(JeCherche)
Do Until isNull(posTrouve)
  posTrouve.CharWeight = com.sun.star.awt.FontWeight.BOLD
  posTrouve = MonDocument.findNext(posTrouve.End, JeCherche)
Loop
```

La structure du descripteur de recherche est un sous-ensemble de celle du descripteur de recherche et remplacement.

L'instruction `findFirst` commence la recherche au début du texte. L'objet *posTrouve* est une zone de texte qui sélectionne la cible trouvée. On peut donc appliquer les méthodes déjà exposées pour modifier cette cible : écrire du texte, changer le style, changer le formatage. Ceci éventuellement après une analyse de la cible (selon son formatage actuel, sa position dans le texte, etc).

L'instruction `findNext` recherche à partir de la position indiquée en premier argument. Comme cet argument est le bord de fin de la cible précédemment trouvée, la recherche reprend là où on l'avait laissée.

Finalement, la recherche échoue et `findNext` (ou `findFirst`) renvoie un élément nul.

Chercher un texte dans une sélection utilisateur

Cette variante suppose que l'utilisateur a sélectionné une seule zone dans le texte. On va rechercher le texte dans cette sélection.

```
Dim MonDocument As Object, MonTexte As Object
Dim CurseurVisible As Object
Dim JeCherche As Object
Dim posTrouve As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
CurseurVisible = MonDocument.currentcontroller.ViewCursor()
JeCherche= MonDocument.createSearchDescriptor

with JeCherche
 .SearchString = "hello"
 .SearchWords = true
end with
```

```
posTrouve = MonDocument.findNext(CurseurVisible.Start, JeCherche)
Do Until isNull(posTrouve)
  if MonTexte.compareRegionEnds(posTrouve, CurseurVisible)< 0 then Exit Do
  posTrouve.CharWeight = com.sun.star.awt.FontWeight.BOLD
  posTrouve = MonDocument.findNext(posTrouve.End, JeCherche)
Loop
```

La recherche commence avec `findNext` afin de démarrer à partir de la position de début de la sélection. La fonction `compareRegionEnds` compare les positions de fin dans les deux curseurs.

Chercher un texte dans une zone quelconque

Pour effectuer la recherche sur une zone quelconque du texte, il suffit d'initialiser un curseur par programmation. Dans cette variante, on recherche dans le deuxième paragraphe du texte :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object
Dim JeCherche As Object
Dim posTrouve As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor ' curseur au début du document
MonCurseur.gotoNextParagraph(false) ' curseur au début du 2eme paragr
MonCurseur.gotoNextParagraph(true) ' curseur sélectionne le 2eme paragr
JeCherche= MonDocument.createSearchDescriptor

with JeCherche
 .SearchString = "hello"
 .SearchWords = true
end with

posTrouve = MonDocument.findNext(MonCurseur.Start, JeCherche)
Do Until isNull(posTrouve)
  if MonTexte.compareRegionEnds(posTrouve, MonCurseur)< 0 then Exit Do
  posTrouve.CharWeight = com.sun.star.awt.FontWeight.BOLD
  posTrouve = MonDocument.findNext(posTrouve.End, JeCherche)
Loop
```

6.8 Les tableaux dans un document Writer

6.8.a Insérer un tableau

En supposant le document Writer ouvert, avec un curseur d'écriture positionné là où vous voulez insérer le tableau, voici comment insérer un tableau de 5 lignes et 9 colonnes :

```
Dim MaTable As Object
MaTable = MonDocument.createInstance("com.sun.star.text.TextTable")
MaTable.initialize(5,9) ' nombre de : lignes, colonnes
MonTexte.insertTextContent(MonCurseur, MaTable, false)
```

L'argument de `createInstance` est une « formule magique » à recopier scrupuleusement.

La méthode `insertTextContent` permet d'insérer un objet quelconque dans le texte, à la position du curseur. Ici, l'objet est un tableau. Le troisième argument signifie :

- false = insérer dans le texte
- true = écraser dans le texte

6.8.b Propriétés du tableau

Le contrôle de la largeur du tableau est traité au chapitre suivant. Principales autres propriétés :

Propriété	*Signification*
`RepeatHeadline`	booléen; true pour répéter la ligne d'en-tête à chaque nouvelle page; valeur false par défaut
`BreakType`	impose un saut de page ou de colonne avant ou après le tableau. Mêmes valeurs que pour le texte ordinaire
`Split`	booléen; false pour imposer que le tableau ne soit pas à cheval sur deux pages ou deux colonnes (à condition que le tableau tienne dans la page !); valeur true par défaut, donc le tableau peut s'étendre sur deux pages ou plus
`BackColor`	couleur du fond de l'ensemble des cellules de la table
`Name`	chaîne de caractères, nom du tableau

Exemple, qui utilise `With` pour alléger l'écriture :

```
MaTable = MonDocument.createInstance("com.sun.star.text.TextTable")
with MaTable
  .RepeatHeadline = true
  .BreakType = com.sun.star.style.BreakType.PAGE_AFTER
  .BackColor = RGB(255,255,204) ' tout en jaune pâle
  .initialize(55,4) ' nombre de : lignes, colonnes
end with
MonTexte.insertTextContent(MonCurseur, MaTable, false)
```

6.8.c Largeur du tableau

Il existe plusieurs propriétés inter-dépendantes qui modifient la largeur d'un tableau. Comme les précédentes propriétés, il est plus clair de les initialiser avant l'appel de `initialize`.

Positionnement horizontal

La propriété `HoriOrient` définit comment le tableau est positionné horizontalement, par rapport aux marges. C'est une horrible constante avec comme valeur par défaut :

```
com.sun.star.text.HoriOrientation.FULL
```

Principales valeurs possibles :

Constante	*Signification*
`NONE`	tableau étalé de la marge de gauche à la marge de droite
`RIGHT`	tableau aligné sur la marge de droite
`LEFT`	tableau aligné sur la marge de gauche
`CENTER`	tableau centré sur l'espace disponible
`FULL`	tableau étalé sur tout l'espace disponible, sans marges de table
`LEFT_AND_WIDTH`	tableau aligné sur la marge de gauche, avec une largeur imposée par la propriété `WIDTH`

Comme la valeur par défaut est `FULL`, il est absolument nécessaire de mettre une autre valeur pour pouvoir imposer une largeur.

Réglages de largeur

Propriété	*Signification*
`IsWidthRelative`	booléen; true si RelativeWidth est à prendre en compte; false si Width est à prendre en compte
`RelativeWidth`	largeur en pourcentage de l'espace libre entre marges de gauche et de droite; valeur entre 1 et 100
`Width`	largeur absolue, en 1/100 de mm
`LeftMargin`	largeur absolue de la marge gauche, en 1/100 de mm
`RightMargin`	largeur absolue de la marge droite, en 1/100 de mm

Applications pratiques

Tableau centré, largeur 80%

```
with MaTable
  .HoriOrient = com.sun.star.text.HoriOrientation.CENTER
  .IsWidthRelative = true
  .RelativeWidth = 80
  .initialize(10,4) ' nombre de : lignes, colonnes
end with
```

Sur OOo version 1.1.0 la largeur de la dernière colonne est incorrecte.
Les propriétés `LeftMargin` et `RightMargin` sont ignorées en orientation `CENTER`.

Tableau centré, largeur absolue

```
with MaTable
  .HoriOrient = com.sun.star.text.HoriOrientation.CENTER
  .IsWidthRelative = false
  .Width = 8000
  .initialize(10,4) ' nombre de : lignes, colonnes
end with
```

Sur OOo version 1.1.0 la largeur obtenue est plus grande que celle indiquée.
Les propriétés `LeftMargin` et `RightMargin` sont ignorées en orientation `CENTER`.

Tableau aligné à gauche, largeur 80%

```
with MaTable
  .HoriOrient = com.sun.star.text.HoriOrientation.LEFT_AND_WIDTH
```

```
  .IsWidthRelative = true
  .RelativeWidth = 80
  .LeftMargin = 20*100 ' 20 mm
  .initialize(10,4) ' nombre de : lignes, colonnes
end with
```

Sur OOo version 1.1.0 la largeur de la dernière colonne est incorrecte.
La propriété `RightMargin` est ignorée en orientation `LEFT_AND_WIDTH`.

Tableau aligné à gauche, largeur absolue

```
with MaTable
  .HoriOrient = com.sun.star.text.HoriOrientation.LEFT_AND_WIDTH
  .IsWidthRelative = false
  .Width = 7000
  .LeftMargin = 10*100 ' 10 mm
  .initialize(10,4) ' nombre de : lignes, colonnes
end with
```

Sur OOo version 1.1.0 la largeur obtenue est plus grande que celle indiquée.
La propriété `RightMargin` est ignorée en orientation `LEFT_AND_WIDTH`.

Tableau étalé entre les marges gauche et droite

```
with MaTable
  .HoriOrient = com.sun.star.text.HoriOrientation.NONE
  .LeftMargin = 20*100 ' 20 mm
  .RightMargin = 35*100 ' 15 mm
  .initialize(10,4) ' nombre de : lignes, colonnes
end with
```

Les propriétés `IsWidthRelative`, `RelativeWidth`, `Width`, sont ignorées.

6.8.d Se déplacer dans un tableau

Pour écrire dans les cellules d'un tableau, il faut savoir comment se positionner sur une des cellules. Dans Writer, la cellule d'un tableau est repérée par son adresse alphanumérique, exemple « B7 », qui est le même repérage que dans une feuille de tableur Calc. Ceci est valable pour des tableaux simples constitués d'une grille régulière de lignes et colonnes. Par exemple, on écrit un texte simple dans une cellule particulière ainsi :

```
Dim UneCellule As Object
UneCellule = MaTable.getCellByName("G3")
UneCellule.String = "un petit texte"
```

Se déplacer avec un curseur de cellule

On peut utiliser un curseur de cellule, notion identique à celle vue avec Calc.

La macro suivante écrit dans toutes les cellules d'un tableau dont on connaît le nombre de lignes et colonnes (9 et 5 dans l'exemple), en allant de gauche à droite sur chaque ligne, et de la première à la dernière ligne :

```
Dim MonCurseurCellule As Object
Dim UneCellule As Object
Dim x1 As Integer, y1 As Integer

MonCurseurCellule = MaTable.createCursorByCellName("A1")
for x1 = 0 to 4
  for y1 = 0 to 8
    UneCellule = MaTable.getCellByName(MonCurseurCellule.RangeName)
    UneCellule.String = Str(x1*100 +y1)
    MonCurseurCellule.goRight(1, false)
  next y1
next x1
```

On définit le curseur de cellule en lui affectant simultanément une adresse initiale de cellule, ici : A1. Ce curseur sera déplacé de cellule en cellule avec la méthode `goRight`, qui descend d'une ligne en fin de ligne précédente.

Une cellule ne peut être obtenue qu'avec son adresse alphanumérique, en argument de `getCellByName`. La propriété `RangeName` la fournit à partir du curseur de cellule.

Modifier la propriété `String` d'une cellule de tableau est la solution la plus simple pour écrire un texte dans la cellule, mais elle fait perdre le formatage par défaut de la cellule. Nous verrons plus loin une méthode plus élaborée.

Liste des instructions pour le curseur

Le deuxième argument de `goRight` signifie :

- false = déplacer le curseur
- true = déplacer le curseur en sélectionnant plusieurs cellules.

Les instructions pour déplacer le curseur ont une syntaxe identique :

Méthode	***Signification***
`goRight(n, false)`	déplacer de n cellules à droite, avec saut de ligne éventuel
`goLeft(n, false)`	déplacer de n cellules à gauche, avec saut de ligne éventuel
`goUp(n, false)`	déplacer de n cellules vers le haut
`goDown(n, false)`	déplacer de n cellules vers le bas
`gotoStart(false)`	déplacer à la première cellule, en haut à gauche
`gotoEnd(false)`	déplacer à la dernière cellule, en bas à droite

Se déplacer sans curseur de cellule

L'utilisation d'un curseur de cellule est assez pénible quand on souhaite accèder à des cellules dont on connaît les coordonnées en ligne et colonne. La macro suivante reprend l'exemple précédent, mais avec une méthode d'accès direct à chaque cellule :

```
Dim UneCellule As Object
Dim x1 As Integer, y1 As Integer

for x1 = 0 to 4    ' index de ligne, style Calc
  for y1 = 0 to 8  ' index de colonne, style Calc
    UneCellule =  MaTable.getCellByName(TextCell(x1, y1))
    UneCellule.String = Str(x1*100 +y1)
  next y1
next x1
```

La fonction *TextCell* n'est pas dans l'API, c'est une fonction que vous pouvez écrire ainsi :

```
rem conversion d'index ligne/colonne vers le nom de la cellule
Function TextCell(Ligne As Integer, Colonne As Integer) As String
if (Colonne > 25) or (Colonne<0) or (Ligne<0) then
  TextCell = "??"
else
  TextCell = Chr(ASC("A") +Colonne) +LTrim(Str(Ligne+1))
end if
End Function
```

Elle convertit une coordonnée numérique ligne/colonne (comptées à partir de zéro) en coordonnée alphanumérique. Elle fonctionne pour des tableaux ayant moins de 26 colonnes, ce qui devrait être suffisant pour vos besoins...

Zone de cellules

Nous pouvons sélectionner plusieurs cellules d'un tableau en déplaçant le curseur de cellule avec l'argument `true`. Exemple :

```
MonCurseurCellule = MaTable.createCursorByCellName("B3")
MonCurseurCellule.goRight(1, true)
MonCurseurCellule.goDown(1, true)
print MonCurseurCellule.Rangename
```

Si vous exécutez la séquence, `print` 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 une zone de cellules. Avec un tel curseur on peut effectuer des opérations communes à toutes les cellules de la zone.

Notez que `createCursorByCellName` ne sait que pointer vers une seule cellule, pas une zone.

6.8.e Ecrire du texte dans une cellule de tableau

Ayant obtenu accès à une cellule par les moyens décrits plus haut, on peut y écrire un texte à l'aide d'un curseur d'écriture à l'intérieur de la cellule. Les possibilités d'écriture sont exactement les mêmes que pour écrire du texte ordinaire. Comme nous l'avons vu, la propriété String remplace tout le contenu d'une cellule par un texte, ou permet de récupérer le texte brut de la cellule; les méthodes `getString` et `setString` n'existent pas.

```
Dim CurseurDansCellule As Object
Dim FinParagraphe As Integer
FinParagraphe = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK
UneCellule =  MaTable.getCellByName("B3") ' (par exemple)

CurseurDansCellule = UneCellule.createTextCursor
UneCellule.insertString(CurseurDansCellule, "OpenOffice est", false)
UneCellule.insertControlCharacter(CurseurDansCellule, FinParagraphe, false)
CurseurDansCellule.CharWeight = com.sun.star.awt.FontWeight.BOLD
UneCellule.insertString(CurseurDansCellule, " formidable !", false)
```

Tout le reste est identique, caractères spéciaux, style, formatage. Reprenez les chapitres correspondants en remplaçant *MonCurseur* par *CurseurDansCellule*.

Si vous avez défini un signet dans une cellule d'un tableau, vous devez récupérer l'objet cellule ainsi :

```
Dim MonDocument As Object, Mazone As Object
Dim CurseurDansCellule As Object, UneCellule As Object

MonDocument = ThisComponent
Mazone = MonDocument.Bookmarks("signet3").getAnchor
UneCellule = Mazone.Cell
CurseurDansCellule = UneCellule.createTextCursorByRange(Mazone)
UneCellule.insertString(CurseurDansCellule, " formidable !", false)
```

6.8.f Somme des cellules d'un tableau

L'exemple suivant va remplir avec des nombres les cellules A2, A3, A4 d'un tableau, puis mettre dans A5 une formule calculant la somme de ces cellules.

```
UneCellule =  MaTable.getCellByName("A2")
UneCellule.Value = 22.5
UneCellule =  MaTable.getCellByName("A3")
UneCellule.Value = -84
UneCellule =  MaTable.getCellByName("A4")
UneCellule.Value = 47/3
UneCellule =  MaTable.getCellByName("A5")
UneCellule.Formula = "sum(<A2:A4>)"
```

Remarque

La propriété `Value` est nécessaire pour remplir la cellule avec une valeur numérique, et non pas un texte. Un nombre comportant des décimales doit être écrit à l'anglaise (avec un point décimal), il sera affiché dans le document texte en respectant la règle de localisation. L'expression 47/3 est calculée par Basic, et son résultat numérique mis dans la propriété `Value` de la cellule.

6.8.g Accéder à une ligne

La propriété `Rows` renvoie la collection des lignes du tableau. On accède à chaque ligne par son numéro d'ordre (ligne zéro pour la première ligne). Cet exemple cite les principales propriétés; il force la hauteur de la 31ème ligneà 20 millimètres, et en colore le fond :

```
Dim LignesCellules As Object

LignesCellules = MaTable.Rows
LignesCellules(30).IsAutoHeight = false
LignesCellules(30).Height = 2000 ' 20 mm
LignesCellules(30).BackColor = RGB(255,255,204)
```

Pour forcer la hauteur de ligne il est nécessaire de désactiver la propriété `IsAutoHeight`, qui par défaut adapter la hauteur au contenu des cellules de la ligne. La hauteur est exprimée en 1/100 mm.

6.8.h Colorer le fond d'un tableau

Tout le tableau

```
MaTable.BackColor = RGB(255,255,204)
```

Une seule cellule

```
UneCellule.BackColor = RGB(255,255,204)
```

Une ligne

```
LignesCellules(0).BackColor = RGB(255,255,204) ' la première ligne
```

Une colonne

Désolé, il n'y a pas de propriété pour cette entité !
Solution : sélectionner la zone de cellules correspondant à la colonne. Exemple pour la colonne D d'un tableau de 5 lignes :

```
MonCurseurCellule = MaTable.createCursorByCellName("D1")
MonCurseurCellule.goDown(4, true)
MonCurseurCellule.BackColor = RGB(255,255,204)
```

6.8.i Insérer plusieurs tableaux

A chaque insertion d'un tableau il est nécessaire d'obtenir un nouveau `TextTable`. Il faut aussi réinitialiser à chaque fois les propriétés du tableau. Dans cet exemple on insère deux tableaux de mêmes dimensions et couleur de fond, dans deux paragraphes du document Writer :

```
MaTable = MonDocument.createInstance("com.sun.star.text.TextTable")
MaTable.initialize(5,4)
MaTable.BackColor = RGB(255,255,204)
MonTexte.insertTextContent(MonCurseur, MaTable, false)

MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(false)

MaTable = MonDocument.createInstance("com.sun.star.text.TextTable")
MaTable.initialize(5,4)
MaTable.BackColor = RGB(255,255,204)
```

```
MonTexte.insertTextContent(MonCurseur, MaTable, false)
```

6.8.j Trouver un tableau par son nom

Vous pouvez simplifier considérablement l'écriture de texte par macro si vous remplissez un document pré-défini (ou un modèle de document) qui contiendra la structure fixe du document, et des tableaux déjà formatés mais vides, qui seront remplis par macro. Rappelons que chaque tableau possède un nom (par défaut : tableau1, tableau2, etc, par ordre de création). Avec le navigateur renommez chaque tableau avec un nom plus spécifique.

Ensuite, il suffit de désigner dans la macro le tableau par son nom :

```
MaTable = MonDocument.TextTables.getByName("jolieTable")
```

Maintenant vous pouvez utiliser toutes les instructions précédentes pour remplir le tableau ou le modifier. Evidemment la macro doit connaître le nom du tableau et le type d'information à y insérer. Le nom du tableau doit être écrit exactement comme dans sa définition.

L'instruction déclenchera une exception s'il n'existe aucun tableau de ce nom. Vous pouvez tester l'existence du tableau ainsi :

```
if MonDocument.TextTables.hasByName("jolieTable") then
  rem le tableau existe
end if
```

Enfin, on renomme un tableau très simplement :

```
MaTable = MonDocument.TextTables.getByName("jolieTable")
MaTable.Name = "MonbeauTableau"
```

Un nom de tableau ne doit pas comporter de caractère espace.

6.8.k Tableaux irréguliers

Prenons comme exemple un tableau initial de 5 lignes et 4 colonnes, bien régulier.

A1	***B1***	***C1***	***D1***
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4
A5	B5	C5	D5

On a indiqué dans chaque cellule sa coordonnée alphanumérique.

Fusionner des cellules

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` :

```
MonCurseurCellule = MaTable.createCursorByCellName("B3")
MonCurseurCellule.goRight(1, true)
MonCurseurCellule.goDown(1, true)
MonCurseurCellule.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 :

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

En fait, OpenOffice 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 alphanumérique de la cellule d'origine, et on lui rajoute 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.

Astuce

Si vous cliquez à l'intérieur d'une cellule de tableau, vous verrez s'afficher ses coordonnées en bas à droite dans la fenêtre Writer ! Très pratique pour s'y retrouver dans les tableaux complexes.

Scinder une cellule

A 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 :

```
MonCurseurCellule.splitRange(2, true) ' scinder horizontalement
```

Le premier argument est le nombre de sous-cellules créées, dans l'exemple c'est 2, donc on se retrouve avec 3 sous-cellules par cellule scindée ! Le deuxième argument vaut :

- `false` pour scinder verticalement
- `true` pour scinder horizontalement

Résultat d'une 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

Résultat d'une 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 pas cohérente avec la précédente, et ne correspond pas à ce qui est dit dans l'API à `getCellByName`. (Ceci est vu sur OOo version 1.0.2 et API daté du 04/02/2003)

Généralisation

Les opérations de scission ou fusion peuvent être répétées sur les nouvelles cellules. Mais l'adressage devient très complexe à gérer.

6.9 Les cadres dans un document Writer

En anglais, le mot « cadre » se traduit par « frame ».

6.9.a Insérer un cadre

Voici comment insérer un cadre, en supposant le document Writer ouvert, avec un curseur d'écriture positionné là où vous voulez insérer le cadre :

```
Dim MonCadre As Object

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)
```

L'argument de `createInstance` est une « formule magique » à recopier scrupuleusement.

La méthode `insertTextContent` permet d'insérer un objet quelconque dans le texte, à la position du curseur. Ici, l'objet est un cadre. Le troisième argument signifie :

- false = insérer dans le texte
- true = écraser dans le texte

L'exemple utilisait le minimum d'instructions. En pratique il faudrait préciser :

1. si la hauteur du cadre s'adapte à son contenu, ou non
2. par rapport à quoi le cadre se positionne (l'ancrage du cadre),
3. sa position absolue par rapport à l'ancre, ou
4. sa position relative définie par :
    i. la zone où peut se placer le cadre,
    ii. sa position horizontale dans cette zone (à gauche, au centre, à droite),
    iii. sa position verticale dans cette zone (en haut, au centre, en bas).

6.9.b Les cadres élastiques

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` :

```
MonCadre.SizeType = com.sun.star.text.SizeType.FIX
```

Les différentes valeurs possibles sont les horribles constantes suivantes :

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)

6.9.c Les différents ancrages de cadre

Le choix du type d'ancrage se fait ainsi :

```
MonCadre.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
```

Les valeurs possibles d'ancrage sont des horribles constantes, listées dans le tableau ci-après.

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; voir exemple plus loin
`AT_CHARACTER`	Ancrage par rapport au caractère pointé par le curseur

6.9.d Concepts utilisés pour le positionnement du 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 : ancrage et alignement. L'alignement correspond au positionnement relatif du cadre.

6.9.e Positionnement absolu du cadre

Positionnement horizontal :

```
MonCadre.HoriOrient = com.sun.star.text.HoriOrientation.NONE
MonCadre.HoriOrientPosition = 2500 ' 25 mm
```

Positionnement vertical :

```
MonCadre.VertOrient = com.sun.star.text.VertOrientation.NONE
MonCadre.VertOrientPosition = 5500 ' 55 mm
```

Les horribles constantes xxx.`NONE` servent à préciser que le positionnement est absolu.

6.9.f Positionnement horizontal relatif

Zone englobante

Cette zone englobante est spécifiée par la propriété `HoriOrientRelation` qui utilise des horribles

constantes, exemple :

```
MonCadre.HoriOrientRelation = com.sun.star.text.RelOrientation.PAGE_LEFT
```

Les valeurs possibles pour `HoriOrientRelation` dépendent du type d'ancrage.

Valeurs possibles pour un ancrage `AT_PARAGRAPH` :

Constante	*Signification*
`FRAME`	Le paragraphe entier, y compris ses marges
`PRINT_AREA`	Le paragraphe entier, sans ses marges (valeur par défaut)
`PAGE_LEFT`	Dans 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`	Dans la marge de droite de la page; même remarque
`FRAME_LEFT`	Dans la marge de gauche du paragraphe; même remarque
`FRAME_RIGHT`	Dans la marge de droite du paragraphe; même remarque

Valeurs possibles pour un ancrage `AT_PAGE` :

Constante	*Signification*
`PAGE_LEFT`	Dans 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`	Dans la marge de droite de la page; même remarque
`PAGE_FRAME`	La page entière, y compris ses marges
`PAGE_PRINT_AREA`	La page entière sans ses marges (valeur par défaut)

Valeurs possibles pour un ancrage `AT_FRAME` :

- comme pour `AT_PARAGRAPH`, mais dans ce cas il s'agit du paragraphe de texte à l'intérieur du premier cadre.

Situation par rapport à la zone englobante

La propriété `HoriOrient` définit comment le cadre est positionné horizontalement, par rapport à la zone englobante. Elle utilise des horribles constantes, exemple :

```
MonCadre.HoriOrient = com.sun.star.text.HoriOrientation.LEFT
```

Les valeurs possibles :

Constante	*Signification*
`RIGHT`	cadre aligné sur la marge de droite
`LEFT`	cadre aligné sur la marge de gauche
`CENTER`	cadre centré sur l'espace disponible

6.9.g Positionnement vertical relatif

Zone englobante

Cette zone englobante est spécifiée par la propriété `VertOrientRelation` qui utilise des horribles

constantes, exemple :

```
MonCadre.VertOrientRelation = com.sun.star.text.RelOrientation.FRAME
```

Les valeurs possibles pour `VertOrientRelation` dépendent du type d'ancrage.

Valeurs possibles pour un ancrage `AT_PARAGRAPH` :

Constante	*Signification*
`FRAME`	Le paragraphe entier, y compris ses marges
`PRINT_AREA`	Le paragraphe entier, sans ses marges (valeur par défaut)

Valeurs possibles pour un ancrage `AT_PAGE` :

Constante	*Signification*
`PAGE_FRAME`	La page entière, y compris ses marges
`PAGE_PRINT_AREA`	La page entière sans ses marges (valeur par défaut)

Valeurs possibles pour un ancrage `AT_FRAME` :

- comme pour `AT_PARAGRAPH`, mais dans ce cas il s'agit du paragraphe de texte à l'intérieur du premier cadre.

Situation par rapport à la zone englobante

La propriété `VertOrient` définit comment le cadre est positionné verticalement, par rapport à la zone englobante. Elle utilise des horribles constantes, exemple :

```
MonCadre.VertOrient = com.sun.star.text.VertOrientation.TOP
```

Valeurs possibles pour une zone englobante `AT_PAGE` :

Constante	*Signification*
`TOP`	cadre en haut de page
`CENTER`	cadre au centre de la page
`BOTTOM`	cadre en bas de la page

Valeurs possibles pour une zone englobante `AT_PARAGRAPH` et orientation relative `PRINT_AREA` :

Constante	*Signification*
`TOP`	le haut du cadre est centré sur le haut de la ligne de texte
`CENTER`	le cadre est centré sur le milieu de la ligne de texte
`BOTTOM`	le bas du cadre est centré sur le bas de la ligne de texte

6.9.h Exemple complet d'insertion de cadre

Dans cet exemple on crée un petit cadre, situé dans la marge gauche de la page, positionné à droite, et à mi-hauteur de la page. Puis on écrit la lettre X dans le cadre.

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object
Dim MonCadre As Object
```

```
MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor
rem - - positionnez le curseur où vous voulez - -

MonCadre = MonDocument.createInstance("com.sun.star.text.TextFrame")


With MonCadre
  .Width = 1000
  .Height = 800
  .SizeType = com.sun.star.text.SizeType.FIX
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
  .VertOrient = com.sun.star.text.VertOrientation.CENTER
  .VertOrientRelation = com.sun.star.text.RelOrientation.PAGE_FRAME
  .HoriOrient = com.sun.star.text.HoriOrientation.RIGHT
  .HoriOrientRelation = com.sun.star.text.RelOrientation.PAGE_LEFT
End With
MonTexte.insertTextContent( MonCurseur, MonCadre, false)
MonCadre.Text.String = "X"
```

6.9.i Ecrire du texte dans un cadre

Toutes les possibilités d'écriture de texte qui existent pour le texte principal sont aussi disponibles pour écrire dans un cadre. Il suffit d'utiliser une variable texte et un curseur d'écriture comme ceci :

```
Dim MonTexte1 As Object, MonCurseur1 As Object

rem - - initialisation et insertion du cadre - -

MonTexte1 = MonCadre.Text
MonCurseur1 = MonTexte1.createTextCursor

MonTexte1.insertString(MonCurseur1, "Texte dans le cadre", false)
```

6.9.j Insérer plusieurs cadres

A chaque insertion d'un cadre il est nécessaire d'obtenir un nouveau `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 de mêmes dimensions, couleur, position, dans deux paragraphes du document Writer :

```
MonCadre = MonDocument.createInstance("com.sun.star.text.TextFrame")
With MonCadre
  .Height = 1500
  .Width = 4000
  .BackColor = RGB(255,200,255)
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
  .HoriOrient = com.sun.star.text.HoriOrientation.NONE
  .HoriOrientPosition = 2500 ' unité : 1/100 mm
End With
MonTexte.insertTextContent( MonCurseur, MonCadre, false)
MonCadre.Text.String = "insertion 1"

MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(false)

MonCadre = MonDocument.createInstance("com.sun.star.text.TextFrame")
With MonCadre
  .Height = 1500
  .Width = 4000
  .BackColor = RGB(255,200,255)
```

```
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
  .HoriOrient = com.sun.star.text.HoriOrientation.NONE
  .HoriOrientPosition = 2500 ' unité : 1/100 mm
End With
MonTexte.insertTextContent( MonCurseur, MonCadre, false)
MonCadre.Text.String = "insertion 2"
```

6.9.k Exemple de cadre dans un cadre

On peut parfaitement positionner un cadre dans un autre cadre, supposé plus grand.

```
Dim MonDocument As Object, MonTexte As Object, MonTexte1 As Object
Dim MonCurseur  As Object, MonCurseur1 As Object
Dim MonCadre1 As Object, MonCadre2 As Object
Dim FinParagraphe As Integer
FinParagraphe = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor
rem - - positionnez le curseur où vous voulez - -

MonCadre1 = MonDocument.createInstance("com.sun.star.text.TextFrame")


With MonCadre1
  .Width = 9400 ' 94 mm largeur
  .Height = 2530 ' 25,3 mm de haut
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
End With

MonTexte.insertTextContent( MonCurseur, MonCadre1, false)
MonTexte1 = MonCadre1.Text
with MonTexte1
  MonCurseur1 = .createTextCursor
  .insertString(MonCurseur1, "Texte dans cadre1", false)
  .insertControlCharacter(MonCurseur1, FinParagraphe, false)
  .insertString(MonCurseur1, "Texte ligne 2", false)
  .insertControlCharacter(MonCurseur1, FinParagraphe, false)
  .insertString(MonCurseur1, "Texte ligne 3", false)
end with

MonCadre2 = MonDocument.createInstance("com.sun.star.text.TextFrame")

With MonCadre2
  .Width = 1000
  .Height = 1000
  .AnchorFrame = MonCadre1
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_FRAME
  .HoriOrient = com.sun.star.text.HoriOrientation.LEFT
End With
MonTexte1.insertTextContent( MonCurseur1, MonCadre2, false)
MonCadre2.Text.String = "X"
```

Les instructions `With` servent à éviter les répétitions de *MonTexte1* ou *MonCadre2*.

On pourrait aussi bien créer un *MonCurseur2* pour insérer des textes entiers, et même insérer un troisième cadre, et ainsi de suite, à condition d'avoir encore de la place sur la page...

6.9.l Couleur du fond du cadre

Par défaut, le cadre hérite de la couleur de fond de la zone d'insertion. Pour imposer une couleur :

```
MonCadre.BackColor = RGB(255,200,255)
```

6.9.m Trouver un cadre par son nom

Vous pouvez simplifier considérablement l'écriture de texte par macro si vous remplissez un

document pré-défini (ou un modèle de document) qui contiendra la structure fixe du document, et des cadres déjà formatés mais vides, qui seront remplis par macro. Rappelons que chaque cadre possède un nom (par défaut : cadre1, cadre2, etc, par ordre de création). Avec le Navigateur renommez chaque cadre avec un nom plus spécifique.

Ensuite, il suffit de désigner dans la macro le cadre par son nom :

```
MonCadre = MonDocument.TextFrames.getByName("CadreBleu")
```

Maintenant vous pouvez utiliser toutes les instructions précédentes pour remplir le cadre ou le modifier. Evidemment la macro doit connaître le nom du cadre et le type d'information à y insérer. Le nom du cadre doit être écrit exactement comme dans sa définition.

L'instruction déclenchera une exception s'il n'existe aucun cadre de ce nom. Vous pouvez tester l'existence du cadre ainsi :

```
if MonDocument.TextFrames.hasByName("CadreBleu") then
  rem le cadre existe
end if
```

Enfin, on renomme un cadre très simplement :

```
MonCadre = MonDocument.TextFrames.getByName("CadreBleu")
MonCadre.Name = "MonBeauCadre"
```

Un nom de cadre ne doit pas comporter de caractère espace.

6.10 Les dessins dans Writer

Ici, on appelle une forme (anglais : shape) un dessin élémentaire, par exemple un rectangle, une ellipse, etc.

6.10.a La page de dessin de Writer

L'API signale que les formes dans Writer apparaissent sur une unique page de dessin (anglais : draw page). Il s'agit des feuilles que l'on peut voir dans un document Draw ou Impress.

En pratique, cette unique page de dessin pour Writer s'étend sur le document entier : vous pouvez insérer des formes sur un nombre quelconque de pages de texte. L'insertion d'une forme directement sur la page de dessin est possible, mais le positionnement est alors obligatoirement ancré à la page, ce qui est très restrictif.

Vous pouvez aussi insérer des dessins à l'intérieur de cadres du document Writer.

Pour obtenir la page de dessin d'un document Writer :

```
Dim PageDessin As Object

PageDessin = MonDocument.DrawPage
```

6.10.b Insérer une forme dessinée

Voici comment insérer une forme rectangulaire, en supposant le document Writer ouvert, avec un curseur d'écriture positionné là où vous voulez insérer la forme :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object
Dim MaForme As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor

Taille1.Width = 1000
Taille1.Height = 800

MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
MaForme.Size = Taille1
MonTexte.insertTextContent( MonCurseur, MaForme, false)
```

Contrairement aux cadres, il n'existe pas de propriété Width ou Height pour une forme. L'instruction `Dim` *Taille1* crée un objet de type Taille (en anglais : Size) qui servira à dimensionner la future forme. Basic ne permet pas de modifier directement la propriété `Size` d'une forme.

L'argument de `createInstance` est une horrible constante qui précise le type de forme dessinée.

Les différentes formes de base sont listées dans le chapitre Dessiner des formes, qui est valable pour toutes les applications.

La méthode `insertTextContent` permet d'insérer un objet quelconque dans le texte, à la position du curseur. Ici, l'objet est une forme (un dessin élémentaire). Le troisième argument signifie :

- false = insérer dans le texte
- true = écraser dans le texte

L'exemple utilisait le minimum d'instructions. En pratique il faudrait préciser :

1. par rapport à quoi la forme se positionne (l'ancrage de la forme),
2. sa position absolue par rapport à l'ancre

6.10.c Les différents ancrages d'une forme

Le choix du type d'ancrage se fait ainsi :

```
MaForme.AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
```

Les valeurs possibles d'ancrage sont des horribles constantes, listées dans le tableau ci-après.

Constante	*Signification*
`AT_PARAGRAPH`	Ancrage par rapport au paragraphe pointé par le curseur, marges comprises
`AS_CHARACTER`	Ancrage comme si la forme était un caractère; la hauteur de la ligne s'adapte à la taille de la forme (valeur par défaut)
`AT_PAGE`	Ancrage par rapport à la page entière avec ses marges `AnchorPageNo` contient le numéro de page; par défaut c'est la page où se trouve le curseur d'écriture.
`AT_CHARACTER`	Ancrage par rapport au caractère pointé par le curseur

6.10.d Positionnement de la forme

Contrairement aux cadres, pour les formes 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.

Bien que les instructions de positionnement absolu d'un cadre soient acceptées, il est préférable d'utiliser une méthode qui fait appel à une variable de position. Exemple :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object
Dim MaForme As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Position1 As New com.sun.star.awt.Point

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor

Taille1.Width = 1000
Taille1.Height = 800
Position1.x = 5500
Position1.y = 12000

MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
With MaForme
  .Size = Taille1
  .Position = Position1
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
End With

MonTexte.insertTextContent( MonCurseur, MaForme, false)
```

6.10.e Insérer plusieurs formes

A chaque insertion d'une forme il est nécessaire d'obtenir un nouveau `xxxShape`, 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 fois la même ellipse à la même position, dans la

page 4 puis dans la page 1 du document Writer :

```
Taille1.Width = 5400
Taille1.Height = 2530
Position1.x = 5500
Position1.y = 12000

MaForme = MonDocument.createInstance("com.sun.star.drawing.EllipseShape")
With MaForme
  .Size = Taille1
  .Position = Position1
  .FillColor = RGB(200,  150, 200)
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
  .AnchorPageNo = 4
End With
MonTexte.insertTextContent( MonCurseur, MaForme, false)
MaForme.String = "insertion 1"

MaForme = MonDocument.createInstance("com.sun.star.drawing.EllipseShape")
With MaForme
  .Size = Taille1
  .Position = Position1
  .FillColor = RGB(200,  150, 200)
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
  .AnchorPageNo = 1
End With
MonTexte.insertTextContent( MonCurseur, MaForme, false)
MaForme.String = "insertion 2"
```

6.10.f Autres fonctionnalités autour de la forme

Les propriétés `SurroundAnchorOnly`, `SurroundContour`, `ContourOutside`, `Opaque`, ne semblent pas avoir une influence sur l'adaptation du texte à la forme, quelque soit l'ancrage.

Les autres fonctionnalités sont identiques pour Writer et les autres applications. Elles sont décrites dans un chapitre commun :

- Propriétés communes
- Dessin de différentes formes
- Ecriture de texte dans la forme
- Donner un nom à une forme

6.11 Les images dans Writer

6.11.a Insérer une image

Note : dans l'API version anglaise, le terme Graphic est utilisé pour désigner une image « bitmap » ou vectorielle. Il n'y a aucun rapport avec le concept français de graphique.

Voici une première méthode d'insertion d'image, qui suffit pour les images que vous n'avez pas à redimensionner, par exemple une petite image comme :

Supposons le document Writer ouvert, avec un curseur d'écriture positionné là où vous voulez insérer l'image :

```
Dim MonImage As Object

MonImage = MonDocument.createInstance("com.sun.star.text.GraphicObject")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\tip.gif")
MonTexte.insertTextContent(MonCurseur, MonImage, false)
```

L'argument de `createInstance` est une « formule magique » à recopier scrupuleusement.

La méthode `insertTextContent` permet d'insérer un objet quelconque dans le texte, à la position du curseur. Ici, l'objet est une image. Le troisième argument signifie :

- false = insérer dans le texte
- true = écraser dans le texte

Remarque

Ces instructions n'insèrent pas l'image dans le document Writer, mais seulement un lien vers l'image. Si vous déplacez, renommez, supprimez le fichier image référencé, le document affichera une case de lien brisé avec l'adresse URL du fichier.

Ceci est une limitation actuelle de l'API.

Pour effectivement insérer dans le document des images insérées par macro, suivre cette procédure :

1. Menu Edition > Liens...
    - sélectionner tous les liens d'images
    - cliquer sur le bouton Déconnecter; confirmer
2. Sauver le document

6.11.b Positionner une image

Les principes de positionnement décrits pour les cadres sont applicables pour les images.

Dans cet exemple on insère une petite image, située dans la marge gauche de la page, positionnée à droite, et à mi-hauteur de la page :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonImage As Object
Dim MonCurseur  As Object

MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor
```

```
MonImage = MonDocument.createInstance("com.sun.star.text.GraphicObject")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\important.gif")

With MonImage
  .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
  .VertOrient = com.sun.star.text.VertOrientation.CENTER
  .VertOrientRelation = com.sun.star.text.RelOrientation.PAGE_FRAME
  .HoriOrient = com.sun.star.text.HoriOrientation.RIGHT
  .HoriOrientRelation = com.sun.star.text.RelOrientation.PAGE_LEFT
End With

MonTexte.insertTextContent(MonCurseur, MonImage, false)
```

6.11.c Dimensionner une image, methode 1

Il est possible de modifier les dimensions de l'image que nous avons insérée dans le document :

```
MonImage.Width = 1800 ' largeur en 1/100 de mm
MonImage.Height = 3000 ' hauteur en 1/100 de mm
```

Ceci déforme l'image car on ne respecte pas les proportions originales. Il existe deux propriétés, `IsSyncHeightToWidth` et `IsSyncWidthToHeight`, qui sembleraient servir à garder les proportions dans un redimensionnement. Hélas, elles ne servent qu'à marquer la coche « Proportionnel » dans le panneau Image de l'interface utilisateur. Pour conserver les proportions on rajoute quelques instructions :

```
Dim taille as object, propW2H as double
taille = MonImage.ActualSize
propW2H = taille.Width / taille.Height
MonImage.Width = MonImage.Height * propW2H
```

La propriété `ActualSize` nous donne les dimensions originales de l'image, ce qui nous permet de connaître ses proportions. Elle nous permet aussi de ramener l'image à ses dimensions initiales :

```
taille = MonImage.ActualSize
MonImage.Size = taille
```

6.11.d Dimensionner une image, méthode 2

Il exsite une deuxième méthode pour insérer une image : en l'insérant sur la page de dessin de Writer, comme un objet de dessin. Ici il est obligatoire de préciser les dimensions. Exemple :

```
Dim MonImage As Object
Dim Taille1 As New com.sun.star.awt.Size

MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonTexte.insertTextContent(MonCurseur, MonImage, false)

Taille1.Width = 6000 ' largeur en 1/100 de mm
Taille1.Height = 6000 ' hauteur en 1/100 de mm
MonImage.Size = Taille1
```

Ces instructions n'insèrent pas l'image dans le document Writer, mais seulement un lien vers l'image. La remarque vue précédemment s'applique aussi ici.

Evidemment, si l'image originale n'a pas les mêmes proportions, l'image sur le document sera déformée. Pour conserver les proportions on rajoute quelques instructions :

```
Dim MonImage As Object, LeBitmap As Object
Dim Proportion As Double
Dim Taille1 As New com.sun.star.awt.Size

MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonTexte.insertTextContent(MonCurseur, MonImage, false)
```

```
LeBitmap = MonImage.GraphicObjectFillBitmap
Taille1 = LeBitMap.Size ' taille en pixels !
Proportion = Taille1.Height / Taille1.Width
Taille1.Width = 6000 ' largeur en 1/100 de mm
Taille1.Height = Taille1.Width * Proportion
MonImage.Size = Taille1
```

Pour dimensionner l'image sans changer les proportions, on récupère les dimensions de l'image en pixels. Ceci n'est possible qu'après l'instruction `insertTextContent`. La proportion calculée sert à fixer la hauteur en fonction de la largeur. Ces deux dimensions sont en 1/100 de mm.

Remarque

Le positionnement relatif ne fonctionne pas avec cette méthode d'insertion d'image. Seul le positionnement absolu fonctionne.

6.11.e Insérer plusieurs images

A chaque insertion d'image il est nécessaire d'obtenir un nouveau `GraphicObject`, même si on insère plusieurs fois la même image. Exemple, avec la méthode simple d'insertion :

```
MonImage = MonDocument.createInstance("com.sun.star.text.GraphicObject")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonTexte.insertTextContent(MonCurseur, MonImage, false)

rem  déplacer le curseur ailleurs dans le texte
MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(false)
MonCurseur.gotoNextParagraph(false)

MonImage = MonDocument.createInstance("com.sun.star.text.GraphicObject")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonTexte.insertTextContent(MonCurseur, MonImage, false)
```

6.11.f Trouver une image par son nom

Pour pouvoir retrouver une image, un bon moyen est de la nommer. Une fois l'image insérée dans le document, il suffit d'utiliser la propriété `Name` :

```
MonImage.Name = "Mon chien"
```

Ensuite, on retrouve l'image avec les méthodes `getByName` et `hasByName` :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonImage As Object, LesImages As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent
LesImages = MonDocument.GraphicObjects

if LesImages.hasByName("Mon chien") then
  MonImage = LesImages.getByName("Mon chien")
  Taille1.Width = 6000 ' largeur en 1/100 de mm
  Taille1.Height = 3000 ' hauteur en 1/100 de mm
  MonImage.Size = Taille1
else
  print "Nom inexistant !"
end if
```

7 Dessin Draw

Ici, on appelle une forme (anglais : shape) un dessin élémentaire, par exemple un rectangle, une ellipse, etc.

7.1 Trouver la page de dessin

7.1.a Une page existante

Chaque page de dessin dans Draw comporte un onglet. Le nom des onglets est par défaut, dans la version localisée française, Page 1, Page 2, etc. Le numéro de page est le rang de l'onglet, de gauche à droite. Si vous déplacez avec la souris une page non renommée, le nom sur l'onglet va changer pour refléter sa position, ainsi que pour les pages suivantes non renommées.

Avec l'API on accède ainsi à une page :

```
Dim MonDocument As Object
Dim UnePage As Object, lesPages As Object

MonDocument = ThisComponent
lesPages = MonDocument.Drawpages
print "Nombre de pages : "; lesPages.Count
UnePage = lesPages(0) ' Première page de dessin
```

`DrawPages` fournit l'ensemble des pages du document. `Count` donne le nombre de pages existantes. Chaque page est accessible par un index.

On aurait pu se passer de la variable *lesPages* en écrivant la dernière ligne :

```
UnePage = MonDocument.Drawpages(0) ' Première page de dessin
```

Attention

Pour l'API les pages sont numérotées à partir de zéro, dans l'ordre affiché par l'interface Draw (toutes les pages, renommées ou non). Ainsi si l'utilisateur change l'ordre des pages, la macro écrira obstinément sur la page de rang n.

Une meilleure méthode consiste à récupérer la page ayant un nom donné :

```
Dim MonDocument As Object
Dim UnePage As Object, lesPages As Object

MonDocument = ThisComponent
lesPages = MonDocument.Drawpages
UnePage = lesPages.getByName("essai-1")
```

On peut tester si une page d'un nom donné existe :

```
if lesPages.hasByName("essai-1") then
  rem  la page existe bien
end if
```

Attention

L'API permet de récupérer le nom de l'onglet avec la propriété `Name` de la page.
Pour les pages non renommées, le nom récupéré est : page1, page2, etc. L'espace a disparu, et le nom commence par une minuscule.

Draw sait si l'onglet est renommé, mais l'API ne fournit pas cette information.

7.1.b Créer, renommer, dupliquer, supprimer une page

L'API concernant les pages de Draw n'est pas aussi élaboré que pour les feuilles de Calc.

Créer une page vierge

On insère la nouvelle page à droite d'une position donnée parmi les pages du document.

```
Dim MonDocument As Object
Dim NouvellePage As Object, lesPages As Object

MonDocument = ThisComponent
lesPages = MonDocument.Drawpages
' insère une page après la deuxième page
NouvellePage = lesPages.insertNewByIndex(1)
```

Dans cet exemple la nouvelle page aura pour index 2, et pour nom Page 3.

Renommer une page

Renommer une page revient à changer la valeur de la propriété `Name` de la page.

```
UnePage = lesPages.getByName("essai-1")
UnePage.Name = "Sommaire"
```

Déplacer une page

Les fonctions de l'API ne permettent pas de modifier l'ordre des pages.

Dupliquer une page

```
Dim MonDocument As Object
Dim UnePage As Object, lesPages As Object
Dim PageDupl As Object

MonDocument = ThisComponent
lesPages = MonDocument.Drawpages
UnePage = lesPages.getByName("essai-1")
PageDupl = MonDocument.Duplicate(UnePage)
```

La nouvelle page est la copie exacte de la page donnée en argument, avec son contenu. L'onglet de cette nouvelle page sera situé à droite de la page de référence, il aura pour nom Page n avec pour n le rang du nouvel onglet. On peut ensuite renommer la page.

Supprimer une page

```
Dim MonDocument As Object
Dim UnePage As Object, lesPages As Object

MonDocument = ThisComponent
lesPages = MonDocument.Drawpages
UnePage = lesPages.getByName("essai-1")
lesPages.remove(UnePage)
```

7.1.c La page visible par l'utilisateur

Cet exemple obtient la page visible par l'utilisateur, affiche le nom de cette page, et rend visible la page « testPage » à la place.

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object, PageUsager As Object

MonDocument = ThisComponent
rem récupérer la page visible
PageUsager = MonDocument.CurrentController.CurrentPage
Print "La page affichée est : "; PageUsager.Name

UnePage = MonDocument.DrawPages.getByName("testPage")
rem changer la page visible
MonDocument.CurrentController.CurrentPage = UnePage
```

7.1.d Propriétés d'une page de dessin

Une page de dessin possède des propriétés, essentiellement :

Propriété	*Signification*
`Name`	chaîne de caractères, le nom de la page affiché sur l'onglet (voir remarque)
`BorderLeft`	Marge de gauche en 1/100 de mm
`BorderRight`	Marge de droite en 1/100 de mm
`BorderTop`	Marge du haut en 1/100 de mm
`BorderBottom`	Marge du bas en 1/100 de mm
`Height`	Hauteur totale de la page en 1/100 de mm
`Width`	Largeur totale de la page en 1/100 de mm
`Number`	(en lecture seulement) Rang de la page dans l'ordre des onglets, numérotation à partir de 1
`Orientation`	Portrait : `com.sun.star.view.PaperOrientation.PORTRAIT` ou Paysage : `com.sun.star.view.PaperOrientation.LANDSCAPE`

Exemple d'utilisation :

```
Dim MonDocument As Object
Dim UnePage As Object

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")

With UnePage
  print "Marges G, D", .BorderLeft , .BorderRight
  print "Marges H, B", .BorderTop, .BorderBottom
  print "Dimensions de la page H, L", .Height, .Width
  print "Rang de la page", .Number
  Select Case .Orientation
    Case com.sun.star.view.PaperOrientation.PORTRAIT
      print "Portrait"
    Case com.sun.star.view.PaperOrientation.LANDSCAPE
      print "Paysage"
  end select
end with
```

7.2 Les dessins dans Draw

7.2.a Insérer une forme dessinée

Cet exemple minimal insère un rectangle dans une page de dessin nommée « essai-1 » :

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("essai-1")
Taille1.Width = 13400 ' 134 mm de large
Taille1.Height = 2530 ' 25,3 mm de haut

MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
With MaForme
  .Size = Taille1
```

```
  .FillColor = RGB(137,  111, 87)
End With
UnePage.add(MaForme)
```

L'instruction `Dim` *Taille1* crée un objet de type Taille (en anglais : Size) qui servira à dimensionner la future forme. Basic ne permet pas de modifier directement la propriété `Size` d'une forme.

L'argument de `createInstance` est une horrible constante qui précise le type de forme dessinée.

Les différentes formes de base sont listées dans le chapitre Dessiner des formes, qui est valable pour toutes les applications.

Une fois le dessin ajouté à la page, on peut changer ses propriétés, listées plus loin.

7.2.b Positionner la forme

L'exemple précédent a positionné la forme tout en haut à gauche dans la page.

Le positionnement fait appel à une variable intermédiaire, qui définit la position d'un point, en coordonnées x et y. Ce sera la position du coin haut-gauche de la forme. Les coordonnées sont exprimées en 1/100 de mm, relativement au coin haut-gauche de la page de dessin.

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Position1 As New com.sun.star.awt.Point

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("essai-1")
Taille1.Width = 13400 ' 134 mm de large
Taille1.Height = 2530 ' 25,3 mm de haut
Position1.x = 2500 ' 25 mm à droite du coin haut-gauche de la page
Position1.y = 5300 ' 53 mm  plus bas que le coin haut-gauche de la page

MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
With MaForme
  .Size = Taille1
  .Position = Position1
  .FillColor = RGB(137,  111, 87)
End With
UnePage.add(MaForme)
```

Note

Si vous faites un clic droit sur une forme, et affichez sa position, les valeurs sont indiquées par rapport aux marges de la page. Alors que le positionnement par macro est relatif à la page entière.

7.2.c Autres fonctionnalités des formes

Les autres fonctionnalités sont identiques pour Draw et les autres applications. Elles sont décrites dans un chapitre commun :

- Propriétés communes
- Dessin de différentes formes
- Ecriture de texte dans la forme
- Donner un nom à une forme

Si vous souhaitez aller plus loin dans la programmation de dessin dans Draw, consultez le site en anglais de Danny Brewer :

http://kosh.datateamsys.com/~danny/OOo/DannysDrawPowerTools

Vous y verrez notamment :

- comment programmer une « tortue » qui dessine en tirant des traits sur la page de dessin
- un dessin animé fait avec des cercles rebondissant sur les marges de la page de dessin
- de nombreux exemples de macros pour le dessin.

7.3 Les images dans Draw

7.3.a Insérer une image

Note : dans l'API version anglaise, le terme Graphic est utilisé pour désigner une image « bitmap » ou vectorielle. Il n'y a aucun rapport avec le concept français de graphique.

Cet exemple insère une image :

```
Dim MonDocument As Object
Dim UnePage As Object
Dim MonImage As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent

UnePage = MonDocument.DrawPages(2) ' page de la troisième feuille Draw
MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")

UnePage.add(MonImage)

Taille1.Width  = 6000 ' largeur en 1/100 de mm
Taille1.Height = 6000 ' largeur en 1/100 de mm
MonImage.Size = Taille1
```

Si l'image originale n'a pas les mêmes proportions, l'image sur le document sera déformée. Nous verrons plus loin comment redimensionner en gardant les proportions originales.

Ces instructions n'insèrent pas l'image dans le document Draw, mais seulement un lien vers l'image. Si vous déplacez, renommez, supprimez le fichier image référencé, le document affichera une case de lien brisé avec l'adresse URL du fichier.

Ceci est une limitation actuelle de l'API.

Pour effectivement insérer dans le document des images insérées par macro, suivre cette procédure :

3. Menu Edition > Liens...
    - sélectionner l'ensemble des liens images
    - cliquer sur le bouton Déconnecter; confirmer
4. Sauver le document.

7.3.b Positionner une image

L'ancrage et le positionnement d'une image et d'une forme sont identiques. Reportez-vous au chapitre concernant le positionnement des formes dans Draw.

7.3.c Dimensionner une image

Pour dimensionner l'image sans changer les proportions, on récupère les dimensions de l'image en pixels. Ceci n'est possible qu'après l'instruction `PageDessin.add(MonImage)`. La proportion

calculée sert à fixer la hauteur en fonction de la largeur. Ces deux dimensions sont en 1/100 de mm.

Cet exemple positionne l'image dans la feuille et redimensionne l'image à une largeur choisie, tout en gardant ses proportions :

```
Dim MonDocument As Object
Dim UnePage As Object
Dim MonImage As Object
Dim LeBitmap As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Position1 As New com.sun.star.awt.Point
Dim Proportion As Double

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages(2) ' page de la troisième feuille Draw
MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
MonImage.GraphicURL = ConvertToURL(" C:\Liste Images\Medor.jpg ")
Position1.x = 2500 ' 25 mm à droite du coin haut-gauche de la page
Position1.y = 5300 ' 53 mm  plus bas que le coin haut-gauche de la page
UnePage.add(MonImage)

LeBitmap = MonImage.GraphicObjectFillBitmap
Taille1 = LeBitMap.Size ' taille en pixels !
Proportion = Taille1.Height / Taille1.Width
Taille1.Width = 6000 ' largeur en 1/100 de mm
Taille1.Height = Taille1.Width * Proportion
MonImage.Size = Taille1
MonImage.Position = Position1
```

7.3.d Insérer plusieurs images

A chaque insertion d'image il est nécessaire d'obtenir un nouveau GraphicObjectShape, même si on insère plusieurs fois la même image, comme dans l'exemple qui suit. Il comporte un sous-programme facilitant le redimensionnement.

```
Sub Inserer2ImagesIdentiques
Dim MonDocument As Object, LesFeuilles As Object
Dim PageDessin As Object
Dim MonImage As Object
Dim Posit1 As New com.sun.star.awt.Point

MonDocument = ThisComponent

PageDessin = MonDocument.DrawPages(2) ' page de la feuille Calc de rang 2
MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
Posit1.x = 3000
Posit1.y = 6000
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonImage.Position = Posit1
PageDessin.add(MonImage)
LargeProp(MonImage, 6000)

MonImage = MonDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
Posit1.x = 10000
Posit1.y = 8000
MonImage.GraphicURL = ConvertToURL("C:\Liste Images\Medor.jpg")
MonImage.Position = Posit1
PageDessin.add(MonImage)
LargeProp(MonImage, 3500)

End Sub


Sub LargeProp( UneImage As Object, Largeur As Long)
Dim LeBitmap As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Proportion As Double
```

```
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
```

7.3.e Trouver une image par son nom

Pour pouvoir retrouver une image, un bon moyen est de la nommer. Une fois l'image insérée dans le document, il suffit d'utiliser la propriété `Name` :

```
MonImage.Name = "Mon chien"
```

Toutes les images (et les formes) d'une page sont accessibles par un index, exemple :

```
UneForme = UnePage(3)
```

Ceci est peu pratique, mais dans Draw il n'existe pas de méthode API pour retrouver une forme par son nom.

Pas de problème, nous allons utiliser la fonction *FindObjectByName* décrite dans un autre chapitre. Avec cette fonction, nous pouvons retrouver l'image et la modifier :

```
Dim MonDocument As Object, MonTexte As Object
Dim MonImage As Object
Dim UnePage As Object
Dim Taille1 As New com.sun.star.awt.Size

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")

MonImage = FindObjectByName(UnePage, "Mon chien")
if IsNull(MonImage) then
  print "Nom inexistant !"
else
  Taille1.Width = 6000 ' largeur en 1/100 de mm
  Taille1.Height = 3000 ' hauteur en 1/100 de mm
  MonImage.Size = Taille1
end if
```

Attention : cette méthode retrouve un objet quelconque sur la page, aussi évitez de donner le même nom à une forme et à une image de la même page.

8 Présentation Impress

8.1 Fonctions identiques à Draw

Les fonctionnalités de Draw, comme la notion de page, les dessins, les images, sont utilisables à l'identique pour une présentation Impress.

8.2 Fonctions spécifiques à Impress

On peut spécifier une action déclenchée par un clic sur un objet de la diapo en cours. Cette action peut consister à exécuter une macro, ce qui donne toutes possibilités.

Quand aux propriétés spécifiques aux formes dans Impress, elles sont relatives aux effets, et seront probablement seulement configurées à la conception des diapos.

9 Les formes : fonctions communes à toutes les applications

9.1 Propriétés des formes

9.1.a Couleur de fond de la forme

Sans instruction particulière, la forme est créée avec une couleur de fond uniforme, dont le coloris est une valeur par défaut. Pour imposer une autre couleur :

```
MaForme.FillColor = RGB(255,200,255)
```

Mais on peut avoir des couleurs non uniformes, ou pas de couleur, selon la valeur de la propriété `FillStyle`.

```
MaForme.FillStyle = com.sun.star.drawing.FillStyle.NONE
```

Les valeurs sont des horribles constantes :

Constante	*Signification*
`NONE`	transparent
`SOLID`	couleur uniforme(valeur par défaut)
`GRADIENT`	dégradé de couleur – consulter l'API pour détails
`HATCH`	quadrillage – consulter l'API pour détails
`BITMAP`	utilisation d'une image – consulter l'API pour détails

Avec une couleur `SOLID` on peut régler la transparence avec la propriété `FillTransparence` :

```
MaForme.FillTransparence = 60 ' pourcentage
```

La valeur est en pourcentage, 0 pour une couleur opaque, 100 pour une couleur totalement transparente.

9.1.b Forme avec une ombre

L'ombre (anglais : shadow) sert à présenter la forme comme flottant au-dessus de la feuille.

```
With MaForme
  .Shadow = true
  .ShadowColor = RGB(100, 100, 100)
  .ShadowXDistance = -250
  .ShadowYDistance = 250
end with
```

Le paramètre `Shadow` est un booléen qui indique qu'une ombre est nécessaire (sans ombre par défaut)

Le paramètre `ShadowColor` précise la couleur de l'ombre (noire par défaut)

Les deux suivants précisent l'extension de l'ombre par rapport à la forme, exprimée en 1/100 de mm. Une valeur négative sur X met l'ombre à gauche, une valeur négative sur Y met l'ombre en bas. On choisit ainsi l'angle de l'éclairage.

9.1.c 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 trigonométrique (le sens inverse des aiguilles d'une montre).

9.1.d Effet parallélogramme

Cet effet transforme un rectangle en parallélogramme.

```
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 à l'inverse du sens trigonométrique (le sens des aiguilles d'une montre). Pour un autre type de forme, l'effet est équivalent.

9.1.e Apparence du trait de contour

Le trait qui trace le contour de la forme a de multiples propriétés. Voici les principales.

Type de trait

```
MaForme.LineStyle = com.sun.star.drawing.LineStyle.DASH
```

Les valeurs sont des horribles constantes :

Constante	*Signification*
`NONE`	le trait est invisible
`SOLID`	trait continu (valeur par défaut)
`DASH`	trait tireté; la forme du tireté dépend de la propriété `LineDash`

La propriété `LineDash` est un élément composite :

Sous-Propriété	*Signification*
`Style`	la forme du tireté (constante `com.sun.star.drawing.DashStyle.xxx`)
`Dots`	nombre entier; nombre de points dans le tireté
`DotLen`	longueur du point en 1/100 de mm
`Dashes`	nombre entier; nombre de tirets dans le tireté
`DashLen`	longueur d'un tiret en 1/100 de mm
`Distance`	distance entre les points en 1/100 de mm

Les différentes formes de tireté :

Constante	*Signification*
`RECT`	un rectangle
`ROUND`	un disque
`RECTRELATIVE`	un rectangle proportionnel à la longueur de la ligne
`ROUNDRELATIVE`	un disque proportionnel à la longueur de la ligne

Un exemple clarifiera peut-être un peu :

```
Dim LesTirets As New com.sun.star.drawing.LineDash

With LesTirets
  .Style = com.sun.star.drawing.DashStyle.RECT
  .Dots = 3
  .DotLen = 50
  .Dashes = 2
  .DashLen = 200
  .Distance = 100
```

```
end with

With MaForme
  rem ( autres initialisations de la forme )
  .LineStyle = com.sun.star.drawing.LineStyle.DASH
  .LineDash = LesTirets
end with
```

Couleur du trait

```
MaForme.LineColor = RGB(250, 0, 0)
```

Epaisseur du trait

```
MaForme.LineWidth = 100
```

L'épaisseur est exprimée en 1/100 de mm.

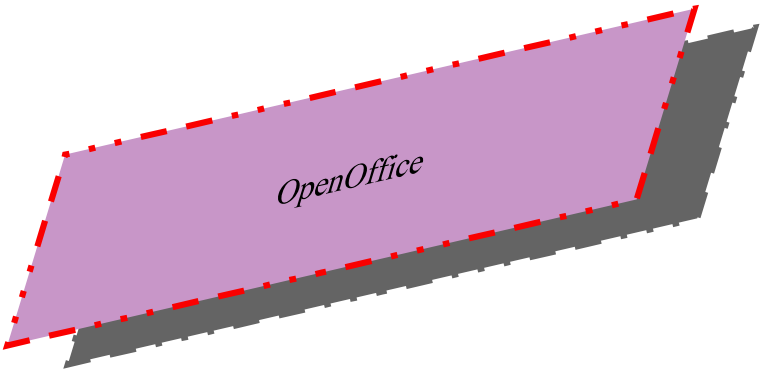
9.1.f Exemple récapitulatif

Cet exemple reprend la plupart des notions de dessin. La forme obtenue est visible en dessous, utiliser un zoom de 200% pour bien voir les détails.

```
Option Explicit

Sub ExemplePourDocIntroAPI
Dim MonDocument As Object, MonTexte As Object
Dim MonCurseur  As Object
Dim MaForme As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Position1 As New com.sun.star.awt.Point
Dim LesTirets As New com.sun.star.drawing.LineDash


MonDocument = ThisComponent
MonTexte = MonDocument.Text
MonCurseur= MonTexte.createTextCursor

MonCurseur = MonDocument.Bookmarks.getByName("ResultatExempleDessin").getAnchor

Taille1.Width = 7400
Taille1.Height = 2530
Position1.x = 4500
Position1.y = 12300
With LesTirets
  .Style = com.sun.star.drawing.DashStyle.RECT
  .Dots = 3
  .DotLen = 50
  .Dashes = 2
  .DashLen = 200
  .Distance = 150
end with

MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
MaForme.Size = Taille1

MonTexte.insertTextContent( MonCurseur, MaForme, false)
With MaForme
  .String = "OpenOffice"
  .Shadow = true
  .ShadowColor = RGB(100, 100, 100) ' gris sombre
  .ShadowXDistance = 800
  .ShadowYDistance = 250
  .RotateAngle = 1500 ' 15 degrés
  .ShearAngle = 3000 ' 30 degrés
  .FillColor = RGB(200,  150, 200)
  .LineStyle = com.sun.star.drawing.LineStyle.DASH
```

```
  .LineDash = LesTirets
  .LineColor = RGB(250, 0, 0) ' rouge
  .LineWidth = 80
'  .FillStyle = com.sun.star.drawing.FillStyle.NONE
end with

End Sub
```

Résultat



9.2 Dessiner des formes

9.2.a Les différentes formes de base

Les principales formes élémentaires disponibles dans `com.sun.star.Drawing` sont :

Forme	*Signification*
`CaptionShape`	Etiquette
`ClosedBezierShape`	Courbe de Bézier fermée (1)
`ConnectorShape`	Connecteur
`EllipseShape`	Ellipse ou cercle
`GraphicObjectShape`	Objet graphique
`LineShape`	Ligne
`MeasureShape`	Ligne de cote
`OpenBezierShape`	Courbe de Bézier ouverte (1)
`PolyLineShape`	Ligne brisée
`PolyPolygonShape`	Polygone
`RectangleShape`	Rectangle ou carré
`TextShape`	Texte

(1) Je ne suis pas arrivé à dessiner ces formes, ni en OOo 1.0.2 ni en 1.1RC.

9.2.b Le rectangle

Déjà vu dans les exemples de base, il se définit par sa taille (hauteur, largeur) et sa position sur la page. Si la hauteur est égale à la largeur, évidemment vous obtenez un carré.

Une propriété spécifique est `CornerRadius` qui sert à arrondir les angles. Sa valeur est le rayon du quart de cercle employé pour les quatre coins.

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Position1 As New com.sun.star.awt.Point

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
Taille1.Width = 13400 ' 134 mm de large
Taille1.Height = 2530 ' 25,3 mm de haut
Position1.x = 2500 ' 25 mm à droite du coin haut-gauche de la page
Position1.y = 5300 ' 53 mm  plus bas que le coin haut-gauche de la page
MaForme = MonDocument.createInstance("com.sun.star.drawing.RectangleShape")
With MaForme
  .Size = Taille1
  .Position = Position1
  .CornerRadius = 250 ' 2,5 mm
  .FillColor = RGB(137,  111, 87)
End With
UnePage.add(MaForme)
```

9.2.c L'ellipse

Déjà vu dans les exemples de base, elle se définit par sa taille (hauteur, largeur) et sa position sur la page. Si la hauteur est égale à la largeur, évidemment vous obtenez un cercle.

Plusieurs propriétés permettent de dessiner des sections d'ellipse.

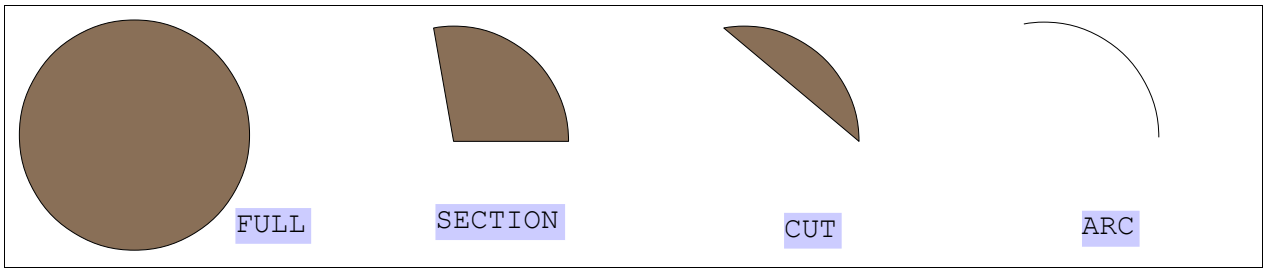
Propriété	***Signification***
`CircleKind`	Type d'ellipse
`CircleStartAngle`	Angle initial de l'arc de la section
`CircleEndAngle`	Angle final de l'arc de la section

Les angles sont mesurés en 1/100 de degré, valeurs positives dans le sens trigonométrique (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. Dans les dessins ci-dessous, l'angle de départ est 0 degré, l'angle final est 100 degrés.

Le type d'ellipse est une horrible constante, dont la valeur par défaut est :

```
com.sun.star.drawing.CircleKind.FULL
```

Avec la valeur `FULL` les propriétés de valeur d'angle ne sont pas significatives. Voici les autres constantes possibles :



Exemple de dessin d'une section de disque :

```
MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
Taille1.Width  = 3000 ' 30 mm de large
Taille1.Height = 3000 ' 30 mm de haut
```

```
Position1.x = 2500 ' 25 mm à droite du coin haut-gauche de la page
Position1.y = 5300 ' 53 mm  plus bas que le coin haut-gauche de la page
MaForme = MonDocument.createInstance("com.sun.star.drawing.EllipseShape")
With MaForme
  .Size = Taille1
  .Position = Position1
  .CircleKind = com.sun.star.drawing.CircleKind.SECTION
  .CircleStartAngle = 0
  .CircleEndAngle = 10000 ' angle en 1/100 de degré
  .FillColor = RGB(137,  111, 87)
End With
UnePage.add(MaForme)
```

9.2.d La ligne simple

Une ligne simple se définit par les coordonnées de ses extrémités. La macro suivante dessine cette ligne :

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim CoordPt(1) As New com.sun.star.awt.Point

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
MaForme = MonDocument.createInstance("com.sun.star.drawing.LineShape")

CoordPt(0).X = 11000 ' position par rapport à la page
CoordPt(0).Y = 8000
CoordPt(1).X = 16000
CoordPt(1).Y = 6000

UnePage.add(MaForme)
MaForme.PolyPolygon = Array(CoordPt())
MaForme.LineColor = RGB(200, 220, 20)
MaForme.LineWidth = 100 ' 1 mm
```

9.2.e La ligne brisée

On décrit la ligne brisée avec les coordonnées de ses points successifs. La description des points doit être remplie après insertion de la forme dans la page. La macro suivante dessine cette ligne brisée :

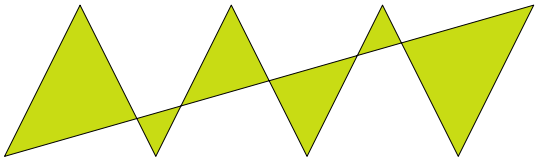
```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim CoordPt(7) As New com.sun.star.awt.Point
Const Y1 = 6000, Y2 = 8000, Ecart = 1000 ' unité 1/100 de mm

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
```

```
MaForme = MonDocument.createInstance("com.sun.star.drawing.PolyLineShape")

CoordPt(0).X = 11000 ' position par rapport à la page
CoordPt(0).Y = Y2
CoordPt(1).X = CoordPt(0).X + Ecart
CoordPt(1).Y = Y1
CoordPt(2).X = CoordPt(1).X + Ecart
CoordPt(2).Y = Y2
CoordPt(3).X = CoordPt(2).X + Ecart
CoordPt(3).Y = Y1
CoordPt(4).X = CoordPt(3).X + Ecart
CoordPt(4).Y = Y2
CoordPt(5).X = CoordPt(4).X + Ecart
CoordPt(5).Y = Y1
CoordPt(6).X = CoordPt(5).X + Ecart
CoordPt(6).Y = Y2
CoordPt(7).X = CoordPt(6).X + Ecart
CoordPt(7).Y = Y1

UnePage.add(MaForme)
MaForme.PolyPolygon = Array(CoordPt())
MaForme.LineColor = RGB(200, 220, 20)
MaForme.LineWidth = 100 ' 1 mm
```

En fait, `LineShape` et `PolyLineShape` sont interchangeables dans ces deux dernières macros.

9.2.f Le Polygone

On décrit le polygone avec les coordonnées de ses points successifs, sachant que le dernier point sera relié automatiquement au premier. La description des points doit être remplie après insertion de la forme dans la page. La macro suivante dessine ce polygone :



```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim CoordPt(7) As New com.sun.star.awt.Point
Const Y1 = 6000, Y2 = 8000, Ecart = 1000 ' unité 1/100 de mm

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
MaForme = MonDocument.createInstance("com.sun.star.drawing.PolyPolygonShape")

CoordPt(0).X = 11000 ' position par rapport à la page
CoordPt(0).Y = Y2
CoordPt(1).X = CoordPt(0).X + Ecart
CoordPt(1).Y = Y1
CoordPt(2).X = CoordPt(1).X + Ecart
CoordPt(2).Y = Y2
CoordPt(3).X = CoordPt(2).X + Ecart
CoordPt(3).Y = Y1
CoordPt(4).X = CoordPt(3).X + Ecart
CoordPt(4).Y = Y2
CoordPt(5).X = CoordPt(4).X + Ecart
CoordPt(5).Y = Y1
CoordPt(6).X = CoordPt(5).X + Ecart
CoordPt(6).Y = Y2
CoordPt(7).X = CoordPt(6).X + Ecart
CoordPt(7).Y = Y1

UnePage.add(MaForme)
```

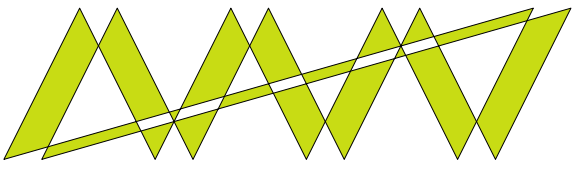
```
MaForme.PolyPolygon = Array(CoordPt())
MaForme.FillColor = RGB(200, 220, 20)
```

9.2.g Combiner plusieurs formes

En réalité, on peut insérer en une seule fois plusieurs formes élémentaires du même type. C'est la raison pour laquelle l'instruction Array(...) a été employée. Le résultat est une seule forme combinée. Comme exemple, reprenons le polygone, et insérons-le deux fois en le décalant horizontalement :

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object
Dim CoordPt(7) As New com.sun.star.awt.Point
Dim CoordPt2(7) As New com.sun.star.awt.Point
Const Y1 = 6000, Y2 = 8000, Ecart = 1000 ' unité 1/100 de mm

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
MaForme = MonDocument.createInstance("com.sun.star.drawing.PolyPolygonShape")

CoordPt(0).X = 11000 ' position par rapport à la page
CoordPt(0).Y = Y2
CoordPt(1).X = CoordPt(0).X + Ecart
CoordPt(1).Y = Y1
CoordPt(2).X = CoordPt(1).X + Ecart
CoordPt(2).Y = Y2
CoordPt(3).X = CoordPt(2).X + Ecart
CoordPt(3).Y = Y1
CoordPt(4).X = CoordPt(3).X + Ecart
CoordPt(4).Y = Y2
CoordPt(5).X = CoordPt(4).X + Ecart
CoordPt(5).Y = Y1
CoordPt(6).X = CoordPt(5).X + Ecart
CoordPt(6).Y = Y2
CoordPt(7).X = CoordPt(6).X + Ecart
CoordPt(7).Y = Y1

CoordPt2(0).X = 11500 ' position par rapport à la page
CoordPt2(0).Y = Y2
CoordPt2(1).X = CoordPt2(0).X + Ecart
CoordPt2(1).Y = Y1
CoordPt2(2).X = CoordPt2(1).X + Ecart
CoordPt2(2).Y = Y2
CoordPt2(3).X = CoordPt2(2).X + Ecart
CoordPt2(3).Y = Y1
CoordPt2(4).X = CoordPt2(3).X + Ecart
CoordPt2(4).Y = Y2
CoordPt2(5).X = CoordPt2(4).X + Ecart
CoordPt2(5).Y = Y1
CoordPt2(6).X = CoordPt2(5).X + Ecart
CoordPt2(6).Y = Y2
CoordPt2(7).X = CoordPt2(6).X + Ecart
CoordPt2(7).Y = Y1

UnePage.add(MaForme)

MaForme.PolyPolygon = Array(CoordPt(), CoordPt2())
MaForme.FillColor = RGB(200, 220, 20)
```

L'image résultante est celle-ci :



Notez l'effet de combinaison, qui « enlève » les surfaces communes aux deux formes élémentaires.

Avec de l'habitude on peut réaliser par macro des dessins précis qui seraient plus difficile de dessiner manuellement avec l'interface graphique. Les créateurs graphiques peuvent y trouver une source d'inspiration illimitée.

9.2.h Disposition relative des formes

Une forme peut en cacher un autre. Tout dépend de l'ordre dans lequel elles sont affichées. Cet ordre est défini par la propriété `ZOrder`, une forme donnée peut cacher toutes celles qui possèdent un `ZOrder` inférieur.

En lisant puis modifiant le `ZOrder` de vos formes vous décidez de l'ordre effectif d'affichage, indépendamment de l'ordre dans lequel vous les avez dessinées.

9.2.i Grouper des formes

La macro ci-dessous regroupe trois formes.

```
Dim MonDocument As Object
Dim MaForme1 As Object, MaForme2 As Object, MaForme3 As Object
Dim UnePage As Object
Dim Taille1 As New com.sun.star.awt.Size
Dim Position1 As New com.sun.star.awt.Point
Dim Collect1 As Object, MonGroupe As Object

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")

Taille1.Width  = 4000
Taille1.Height = 4000
Position1.x = 3000
Position1.y = 5300
MaForme1 = MonDocument.createInstance("com.sun.star.drawing.EllipseShape")
MaForme1.Size = Taille1
MaForme1.Position = Position1
UnePage.add(MaForme1)

Position1.x = 9000
Position1.y = 5300
MaForme2 = MonDocument.createInstance("com.sun.star.drawing.EllipseShape")
MaForme2.Size = Taille1
MaForme2.Position = Position1
UnePage.add(MaForme2)

Taille1.Width  = 5600
Taille1.Height = 8000
Position1.x = 5200
Position1.y = 6500
MaForme3 = MonDocument.createInstance("com.sun.star.drawing.EllipseShape")
MaForme3.Size = Taille1
MaForme3.Position = Position1
UnePage.add(MaForme3)

Collect1 = createUnoService("com.sun.star.drawing.ShapeCollection")
Collect1.Add(MaForme1)
Collect1.Add(MaForme2)
Collect1.Add(MaForme3)
MonGroupe = UnePage.group(Collect1)
```

Il est ensuite possible de déplacer ou redimensionner la forme composite résultante, 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.

9.3 Ecrire du texte dans la forme

Une fois la forme insérée avec l'instruction `Add`, un simple texte s'écrit ainsi :

```
MaForme.String = "OpenOffice"
```

Cette manière de faire ne nous permet pas de choisir le format du texte. Pour obtenir des posibilités plus étendues, on utilise les mêmes fonctionnalités de texte que Writer. Les instructions suivantes écrivent deux lignes dans la forme, en choisissant la taille des caractères la graisse et en changeant la police courante; elles seront claires après avoir lu les premiers chapitres sur Writer.

```
Dim MonTexte1 As Object, MonCurseur1 As Object
Dim FinParagraphe As Integer

FinParagraphe = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK

rem - initialisation et insertion de la forme, voir plus haut -

MonTexte1 = MaForme.Text
MonCurseur1 = MaForme.createTextCursor
MonCurseur1.CharHeight = 16
MonTexte1 = MaForme.Text
MonTexte1.InsertString(MonCurseur1, "OpenOffice", false)
MonTexte1.InsertControlCharacter(MonCurseur1, FinParagraphe, false)
MonCurseur1.CharFontName = "Arial"
MonCurseur1.CharHeight = 24
MonCurseur1.CharWeight = com.sun.star.awt.FontWeight.BOLD
MonTexte1.InsertString(MonCurseur1, "En gras, police Arial", false)
```

9.4 Trouver une forme

9.4.a Trouver les formes sélectionnées par l'utilisateur

L'utilisateur peut sélectionner une ou plusieurs formes. La propriété `CurrentSelection` fournit une liste des formes et images sélectionnées dans la page. Mais attention, si rien n'est sélectionné, cette propriété ne peut être utilisée; il faut tester son existence avant !

Cet exemple balaie les formes sélectionnées et impose une couleur de remplissage. Cela ne marchera évidemment que pour des formes qui le permettent.

```
Dim MaForme As Object, LesFormes As Object
Dim UnePage As Object
Dim s1 As Long

MonDocument = ThisComponent
UnePage = MonDocument.CurrentController.CurrentPage
if isObject(MonDocument.CurrentController.Selection) then
  LesFormes = MonDocument.CurrentSelection
  for s1 = 0 to LesFormes.Count -1
    MaForme = LesFormes(s1)
    MaForme.FillColor = RGB(100,200, 100)
  next
end if
```

9.4.b Nommer une forme

Pour pouvoir retrouver une forme, un bon moyen est de la nommer. Une fois la forme insérée dans le document, il suffit d'utiliser la propriété `Name` :

```
MaForme.Name = "Premier Objet"
```

Attention : vous pouvez ainsi donner le même nom à plusieurs objets de la même page !

En version 1.0.x, l'API a un avantage sur l'interface usager, qui ne permet pas de renommer une

forme simple. En version 1.1, cette limite est levée.

Après exécution le Navigateur vous affichera les objets nommés dans la page; éventuellement faire un double clic sur l'icône de la page, dans le Navigateur.

9.4.c Trouver une forme par son nom

Toutes les formes (et les images) d'une page sont accessibles par un index, exemple :

```
UneForme = UnePage(3)
```

Ceci est peu pratique, mais il n'existe pas de méthode API pour retrouver une forme par son nom. Pas de problème, nous allons écrire une fonction très générale, inspirée par le remarquable travail de Danny Brewer, voir son site web (en anglais) :

http://kosh.datateamsys.com/~danny/OOo/DannysDrawPowerTools

```
Function FindObjectByName(LaPage As Object, NomForme As String ) As Object
Dim formeX As Object
Dim f1 As Long
  For f1 = 0 To LaPage.Count - 1
    formeX = LaPage(f1)
    If formeX.Name = NomForme Then
      FindObjectByName = formeX
      Exit Function
    EndIf
  Next
rem ici la fonction retourne null
End Function
```

Maintenant nous pouvons retrouver la forme et la modifier :

```
rem  - exemple pour un document Draw -
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
MaForme = FindObjectByName(UnePage, "Premier Objet")
if IsNull(MaForme) then
  print "Nom inexistant !"
else
  MaForme.FillColor = RGB(207,  150, 180)
end if
```

Attention : cette méthode retrouve un objet quelconque sur la page, aussi évitez de donner le même nom à une forme et à une image de la même page.

9.4.d Sélectionner visiblement une forme

Nous allons sélectionner une forme que nous avons trouvée par son nom. Cette macro est écrite pour Draw; on suppose que cette forme est dans une autre page que celle qui est affichée.

```
Dim MonDocument As Object
Dim MaForme As Object
Dim UnePage As Object

MonDocument = ThisComponent
UnePage = MonDocument.DrawPages.getByName("testPage")
MaForme = FindObjectByName(UnePage, "Premier Objet")
if IsNull(MaForme) then
  print "Nom inexistant !"
else
  MonDocument.CurrentController.CurrentPage = UnePage
  MonDocument.CurrentController.Select(MaForme)
```

```
end if
```

Il est nécessaire, comme nous l'avons fait, d'afficher la page de dessin où se trouve la forme, sinon la sélection ne se fera pas.

10 Informations annexes

10.1 Basic simplifie l'utilisation de l'API

Si vous lisez dans des forums ou d'autres documents des macros Basic utilisant l'API, vous verrez des constructions un peu différentes de celles décrites ici. La raison est qu'il y a plusieurs manières de faire la même chose en Basic, et certaines macros sont inspirées du langage Java, qui n'accepte que certaines constructions plus lourdes.

10.1.a Les parenthèses optionnelles

Quand une méthode ne comporte aucun argument, les parenthèses ne sont pas nécessaires.

```
MonDocument.Store()
```

Simplification Basic

```
MonDocument.Store
```

10.1.b Les getXxxx et setXxxx

L'API utilise abondamment des méthodes en `getXxxx` pour obtenir une information, et en `setXxxx` pour modifier l'information.

Basic fournit des **pseudo**-propriétés sous forme d'un `Xxxx` accessible en lecture et/ou écriture. Ecriture utilisant strictement l'API :

```
tx3 = UneCellule.getString()
UneCellule.setString("Bonsoir")
```

Simplification Basic :

```
tx3 = UneCellule.String
UneCellule.String = "Bonsoir"
```

`String` n'est pas une véritable propriété de cellule, et n'est pas décrit dans la documentation API. C'est une facilité offerte par le Basic OpenOffice.

10.1.c L'accès aux propriétés

La plupart des propriétés sont accessibles en écriture et en lecture. Cela veut dire qu'on peut en général modifier ou récupérer la valeur actuelle d'une propriété. Exemple :

```
couleur = UneCellule.CellBackColor          ' récupérer la valeur actuelle
UneCellule.CellBackColor = RGB(255,255,204) ' affecter une valeur
```

Vous pourrez rencontrer dans la littérature une forme plus complexe mais équivalente pour manipuler une propriété :

```
couleur =  UneCellule.getPropertyValue("CellBackColor")
UneCellule.setPropertyValue("CellBackColor", RGB(255,255,204))
```

Dans cette tournure le nom de la propriété doit respecter la casse (majuscules, minuscules).

10.2 Qu'est-ce qu'un mot ?

Un mot est une séquence de caractères entre deux séparateurs de mots ! Bien, mais qu'est-ce qu'un séparateur de mot ? C'est soit un espace, un caractère de fin de paragraphe ou de fin de ligne, ou un caractère parmi ceux de la chaîne de caractères `WordSeparator` qui est une propriété du document.

Le nombre de mots d'un document est dans sa propriété `WordCount`.

La notion de mot est un peu différente pour la gestion de curseur. Vous vous en apercevrez si

vous déplacez le curseur visible en tapant Control+ touche de déplacement à droite. Le curseur bute sur des caractères comme « + / , ». Ce sera aussi le cas avec `gotoNextWord`.

La propriété de document `WordCount` effectue un calcul de mots sophistiqué qui n'est pas accessible comme routine indépendante. Pour la version 1.1 d'OpenOffice.org, Andrew Brown a créé une macro donnant un résultat équivalent. Vous la trouverez avec d'autres macros intéressantes sous le nom de "Andrew Brown's macros" sur le site OOoForum. La version 2 d'OpenOffice.org permet de compter les mots d'une sélection.

10.3 Les horribles constantes

L'API fait un large usage de constantes ou des valeurs d'énumération; ce sont en fait des valeurs numériques simples comme 1, 2, 3, 4. Vous ne devez pas utiliser ces nombres, car ils peuvent évoluer avec l'implémentation. Par contre les noms de ces constantes sont plus stables.

Mais le même concept peut donner lieu à des constantes dans différents modules. Par exemple vous pouvez rechercher dans l'index de l'API les définitions de `LEFT` : environ 26 définitions indépendantes, avec des valeurs réelles souvent très différentes.

Pour distinguer les constantes relatives à deux modules différents, il est nécessaire de qualifier le nom d'une constante avec le nom complet du module, qui est défini hiérarchiquement.

Cette hiérarchie est aussi celle de l'arborescence des pages HTML de l'API. Lorsque vous arrivez sur une page décrivant les valeurs d'une énumération vous aurez quelque chose comme :

Adresse de la page HTML : blabla/com/sun/star/table/CellHoriJustify.html

Au début de cette page HTML vous verrez :

:: com :: sun :: star :: table ::

enum CellHoriJustify

Et différentes valeurs comme STANDARD LEFT CENTER etc.

La règle de transcription vers une constante Basic est facilement déduite de cet exemple :

```
Dim maJustif As Integer
rem Utilisation d'une variable intermédiaire pour faciliter la lecture
maJustif = com.sun.star.table.CellHoriJustify.STANDARD
UneCellule.HoriJustify = maJustif

rem Utilisation directe de la constante
UneCellule.HoriJustify = com.sun.star.table.CellHoriJustify.STANDARD
```

Attention

Le nom complet de la constante doit être exactement recopié, majuscules et minuscules.

10.4 Les Couleurs

Dans OpenOffice une couleur est définie par trois valeurs entières , chacune entre 0 et 255, qui représentent l'intensité des couleurs primaires Rouge, Vert, Bleu (en anglais : `Red`, `Green`, `Blue`).

Chaque couleur est représentable dans un octet, les trois octets étant combinés sous la forme d'un entier long (type de donnée : `Long`).

Connaissant les valeurs de chaque couleur primaire on peut déterminer la valeur OpenOffice de la couleur résultante :

```
Dim couleur As Long
Dim partRouge As Integer, partVerte As Integer, partBleue As Integer

partRouge = 120
partBleue = 60
```

```
partVerte = 220
couleur = RGB(partRouge, partVerte, partBleue)
```

Connaissant la valeur de couleur on peut retrouver la valeur de chaque couleur primaire :

```
Dim couleur As Long
Dim partRouge As Integer, partVerte As Integer, partBleue As Integer

couleur = 8253534
partBleue = Blue(couleur)
partVerte = Green(couleur)
partRouge = Red(couleur)
```

Astuce

L'outil de configuration de palette de couleurs, accessible via le menu Outils > Options > OpenOffice.org > Couleurs vous permet de trouver le triplet correspondant à la couleur de votre choix. Cliquez sur le bouton Editer, cliquez sur la couleur de votre choix dans l'arc-en-ciel, recopiez sur un bout de papier les valeurs Rouge-Vert-Bleu qui apparaissent, et fermez la fenêtre par Annuler.

10.5 Consulter l'API

Les procédures et fonctions décrites ici ne sont qu'un infime sous-ensemble de celles disponibles. Pour en savoir plus, il faut consulter la documentation de l'API . Mais elle est immense et déroutante, difficile à exploiter, et **en anglais seulement**. Il est fortement conseillé de lire au préalable le Guide du Programmeur de Sun, ou le livre Programmation OpenOffice.org (voir chapitre 1.2).

L'outil **Xray**, disponible sur la page des HowTo, section Programmation Basic, du site web fr.OpenOffice.org, vous permettra de connaître les possibilités des objets API manipulés dans vos macros. Avec un peu d'habitude, il devient indispensable.

Le SDK est disponible sur le site http://api.openoffice.org/ et plus précisément à la page http://www.openoffice.org/dev_docs/source/sdk/ .
La version OpenOffice.org SDK 1.1 disponible à la date de ce document pèse 40,5 Moctets pour Windows, 27,9 Moctets pour Linux et 29,2 Moctets pour Solaris.

Il existe aussi une version préliminaire du SDK pour OOo 2.0, qui contient une description plus à jour de certaines fonctionalités introduites à partir de OOo 1.1.2.

Le SDK se décompose en une immense arborescence qui comprend l'API, le Developer's Guide, et de nombreux autres documents.

Le Developer'Guide est disponible dans le SDK en format PDF (un pavé de 1000 pages indigestes) et sous forme HTML très pratique à consulter mais lourdes à charger, avec des liens vers la documentation API, elle-même en HTML. Vous trouverez dans le Developer'Guide des informations sur la « philosophie » de l'API et deux chapitres sur son implémentation avec Basic. En pratique le Developer'Guide est seulement accessible à des spécialistes informaticiens; les exemples sont en général fournis en Java.

L'API seule est seulement disponible sous forme de plus de 4000 pages HTML, consultables en ligne . Les liens sont nombreux entre les différents éléments, et un bon navigateur est nécessaire (par exemple Opera, ou Firefox, qui facilitent l'ouverture de fenêtres simultanées). Les pages disponibles en ligne ont une présentation un peu différente de celles du fichier compressé.

Utilisez les pages d'index pour sauter assez rapidement à ce qui vous intéresse, et pour repérer des éléments nouveaux.

On retrouvera les mêmes noms de procédures ou fonctions dans différents contextes, avec des fonctionnalités variant légèrement.

Le tableau de la page suivant fournit quelques repères.

.

Sujet	*Où trouver l'information*
Document	com > sun > star >Document
Texte	com > sun > star > text
Tableur	com > sun > star > sheet
Diagramme de données	com > sun > star >chart
Dessin	com > sun > star > drawing
Présentation (Impress)	com > sun > star > presentation
Styles de paragraphe et de cellule	com > sun > star > style
Tableaux, en général (Calc, Writer)	com > sun > star > table
Propriété de ligne de tableau (Writer)	index > TextTableRow
Contenu de Cellule	index > XCell
Formatage de cellule	index > CellProperties (table pour Calc) index > CellProperties (text pour Writer)
Méthodes pour zone de cellules	index > XCellRange
Zone de texte	index > XTextRange TextRangeCompare XTextRangeMover
Curseur pour aller de cellule en cellule	index > XCellCursor (pour Calc) index > XTextTableCursor (pour Writer)
Propriétés d'un cadre de texte	index > TextFrame > BaseFrame
Propriétés de paragraphe et de caractère dans un texte	com > sun > star > text > Paragraph
Propriétés de paragraphe de texte	index > ParagraphProperties
Forme (tableau, cadre, dessins)	index > Shape
Propriétés de trait (dessin)	index > LineProperties
Remplissage d'une forme (couleur de fond)	index > FillProperties

11 Crédits

Auteur: Bernard Marcelly

Remerciements:

Intégré par : Sophie Gautier

Dernière modification: 03 Novembre 2005

Contacts : Projet Documentation OpenOffice.org http://fr.openoffice.org

Traduction :

12 Licence

Appendix

Public Documentation License Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the "License"); you may only use this Documentation if you comply with the terms of this License. A copy of the License is available at *http://www.openoffice.org/licenses/PDL.html*.

The Original Documentation is : L'API OpenOffice.org (presque) sans peine
The Initial Writer of the Original Documentation is Bernard Marcelly
 (Initial Writer contact(s): marcelly@club-internet.fr)

Contributor(s): _____.
Portions created by _____ are Copyright (C)_____*[Insert year(s)]*.
All Rights Reserved. (Contributor contact(s):_____*[Insert hyperlink/alias]*).

NOTE: The text of this **Appendix** may differ slightly from the text of the notices in the files of the Original Documentation. You should use the text of this **Appendix** rather than the text found in the Original Documentation for Your Modifications.