A

Comprendre l'API d'OpenOffice.org

Cette annexe offre une introduction fort utile à la principale source d'information officielle sur les mécanismes internes à OpenOffice.org. Tous ces documents, en anglais, sont accessibles à un utilisateur confirmé.

Qu'est-ce que l'API ?

L'API (*Application Programming Interface*) d'OpenOffice.org est un ensemble de points d'entrée permettant de manipuler OpenOffice.org – sans en couvrir tous les mécanismes. Sans être liée à un langage de programmation particulier, elle est accessible de manière privilégiée avec OOoBasic, mais aussi avec des langages tels que Java, Python, Delphi, voire d'autres outils de script tel VBscript.

L'API est un système logiciel très complexe par son étendue et par ses concepts ; elle est composée de très nombreux objets héritant les uns des autres. La documentation des objets de l'API est organisée en un arbre dont la racine est :

```
com.sun.star.
```

De cette racine, elle se subdivise en modules, qui sont des groupements de programmes, par exemple :

```
com.sun.star.text.
```

Tout élément de l'API se situe à un nœud de cet arbre, en juxtaposant les branches successives :

```
com.sun.star.text.WrapTextMode.PARALLEL
```

Un module se décompose en un service ou plusieurs, et parfois en sous-modules, comme pour le module `text`.

Un service peut lui-même comporter des services (en anglais *included services*). Il peut exposer des propriétés. Ce sont des « vraies » propriétés (nous verrons plus loin qu'OOoBasic présente aussi des pseudo-propriétés). Un service possède en général une ou plusieurs interfaces. La documentation dit que le service exporte des interfaces, ce qui signifie qu'il les met à la disposition du programmeur.

Une interface contient des méthodes. Ce sont des sous-programmes utilisables à partir de l'objet considéré. Les méthodes utilisent des arguments et renvoient éventuellement un résultat (sous-programme fonction). Certaines interfaces contiennent des attributs ; un attribut s'utilise comme une propriété d'un service.

Chaque argument et chaque résultat d'une méthode, chaque propriété et chaque attribut peuvent être une donnée simple ou complexe :

- une donnée d'un type simple (booléen, entier, flottant) ;
- une constante nommée, qui est un type entier dont les valeurs possibles sont définies avec des noms qualifiés ;
- un objet API, donnant accès à des services, propriétés, interfaces ;
- une séquence de données (elle apparaît dans les langages de programmation sous forme d'un tableau à une dimension) ;
- une structure de données, qui est un regroupement de données accessibles individuellement ; chaque élément de la structure pouvant être une donnée simple ou complexe.

Comme l'API est indépendante du langage, le type simple indiqué pour une donnée doit être « traduit » dans le type de données le plus proche pour le langage de programmation utilisé. Cela peut poser quelques difficultés : par exemple, OOoBasic ne possède pas de type `hyper`.

Un objet API, en dehors des constantes et des structures, comporte un ou plusieurs services. Il y a lieu de distinguer les services pris en charge, directement utilisables, et les services disponibles, qui peuvent être invoqués avec la méthode `createInstance` de l'objet.

Règles typographiques des noms dans l'API

Les noms employés dans l'API suivent quelques règles d'homogénéité :

- Un nom de propriété ou d'attribut débute toujours par une lettre majuscule : `CharFontName`.
- Un nom d'interface débute par un X majuscule.
- Un nom de méthode débute en principe par un verbe (en anglais), commençant par une lettre minuscule : `loadComponentFromURL`.
- Si un nom est composé de plusieurs mots, les mots suivant le premier débutent par une majuscule (sauf le premier) : `createReplaceDescriptor`, `CharFontName`.
- Le dernier terme du nom d'une constante est entièrement en majuscules : `com.sun.star.sheet.Border.RIGHT`.

L'API réelle et l'API selon OOoBasic

OOoBasic a plusieurs avantages par rapport aux autres langages :

- Il est le plus intégré à l'application.
- Il est facile à apprendre.
- Il est conçu pour simplifier l'accès aux primitives de l'API : il n'impose pas de respecter les majuscules et minuscules dans l'emploi des noms de routines et de propriétés.
- Il connaît les valeurs des constantes nommées.
- Il permet d'utiliser directement les interfaces d'un objet, contrairement à C++, Java™ ou ses dérivés.
- Il permet d'utiliser comme une pseudo-propriété le couple de méthodes `get` et `set` manipulant la même donnée interne, contrairement à Java.

Pour toutes ces raisons, les exemples en Java, nombreux dans la documentation, sont bien plus difficiles à lire que leurs équivalents en OOoBasic. À titre d'exemple, voici un extrait de code en Java qui modifie un mot dans un texte Writer et modifie le curseur pour écrire en gras.

```
mxDocCursor = mxDocText.createTextCursor();
XSentenceCursor xSentenceCursor = (XSentenceCursor)
UnoRuntime.queryInterface(XSentenceCursor.class, mxDocCursor);
xSentenceCursor.gotoNextSentence(false);
XWordCursor xWordCursor = (XWordCursor)
UnoRuntime.queryInterface(XWordCursor.class, mxDocCursor);
xWordCursor.gotoNextWord(false);
xWordCursor.gotoNextWord(true);
mxDocText.insertString(xWordCursor, "hello ", true);
```

```
XPropertySet xCursorProps = (XPropertySet)
UnoRuntime.queryInterface(XPropertySet.class, mxDocCursor);
xCursorProps.setPropertyValue("CharWeight",
new Float(com.sun.star.awt.FontWeight.BOLD))
```

Ce code Java nécessite trois variables supplémentaires pour gérer le curseur, une pour chaque interface nécessaire. Voici pour comparaison le code OOoBasic équivalent, qui n'utilise que la variable curseur :

```
mxDocCursor = mxDocText.createTextCursor
mxDocCursor.gotoNextSentence(false)
mxDocCursor.gotoNextWord(false)
mxDocCursor.gotoNextWord(true)
mxDocText.insertString(mxDocCursor, "hello ", true)
mxDocCursor.CharWeight = com.sun.star.awt.FontWeight.BOLD
```

Lorsqu'il existe deux méthodes complémentaires simples, l'une servant à affecter une valeur à une donnée interne, l'autre servant à obtenir la valeur de cette donnée, OOoBasic combine les deux sous la forme d'une pseudo-propriété, utilisable comme une simple variable. Voici deux codages OOoBasic qui réalisent exactement la même chose :

```
Dim Taille As Object
' codage autorisé par OOoBasic
Taille = dessin.Size
Taille.Width = Taille.Width * 2
dessin.Size = Taille

' application stricte de l'API
Taille = dessin.getSize()
Taille.Width = Taille.Width * 2
dessin.setSize(Taille)
```

L'utilisateur OOoBasic ne voit qu'une propriété `Size`. En réalité, cette propriété n'existe pas, mais elle est un raccourci vers deux méthodes :

- `setSize(`valeur`)` qui modifie la taille (en anglais *size*) de l'objet ;
- `getSize()` qui renvoie la taille de l'objet.

Pour retrouver dans l'API la description d'une propriété d'objet, il est donc nécessaire d'ajouter `get` ou `set` s'il s'agit d'une pseudo-propriété. Certaines données internes `Xyz` peuvent être manipulées seulement par `getXyz`, ou seulement par `setXyz`. Dans ces cas, la pseudo-propriété est restreinte à la seule lecture ou écriture.

On accède aux vraies propriétés des objets de l'API de manière plus simple en OOoBasic que dans d'autres langages comme Java. Exemple :

```
' codage autorisé par OOoBasic
couleur = UneCellule.CellBackColor
UneCellule.CellBackColor = RGB(255,255,204)

' deuxième manière, plus complexe, et aussi valide en OOoBasic
' ici la casse de CellBackColor doit être respectée
couleur = UneCellule.getPropertyValue("CellBackColor")
UneCellule.setPropertyValue("CellBackColor", RGB(255,255,204))
```

Un objet dans une collection est accessible en OOoBasic par une indexation, comme si la collection était un tableau. En réalité, OOoBasic fait appel à la fonction `getByIndex` de la collection :

```
' codage autorisé par OOoBasic
uneFeuille = monDocument.Sheets(1)

' deuxième manière, plus complexe, et aussi valide en OOoBasic
uneFeuille = monDocument.Sheets.getByIndex(1)
```

Il existe aussi un raccourci OOoBasic pour accéder par son nom à un objet de collection. Nous ne l'avons pas employé car il peut donner des expressions ambiguës.

```
' codage recommandé
uneFeuille = monDocument.Sheets.getByName("Total")

' deuxième manière, valide en OOoBasic, déconseillée
uneFeuille = monDocument.Sheets.Total
```

> ALLER PLUS LOIN **Transcription entre UNO et un langage de programmation**
>
> La technologie UNO implique certains concepts de programmation (types de données, gestion des erreurs, accès aux objets). La mise en équivalence pour un langage de programmation est appelée *binding*. Le *Developer's Guide* décrit au chapitre *Professional UNO>UNO language bindings* les règles de transcription pour les langages Java, C++, Basic, CLI, et l'interface COM Automation.

Les fonctions Basic dédiées à l'API

Tout au long de cet ouvrage, nous avons utilisé plusieurs fonctions OOoBasic qui facilitent l'accès à l'API. En voici une liste plus systématique.

GetProcessServiceManager

L'objet logiciel obtenu par `GetProcessServiceManager` est, en quelque sorte, la mère de tous les objets de l'API. Il permet d'obtenir un service initialisé, soit par défaut, soit avec des arguments. Le premier cas est réalisé plus simplement avec `CreateUnoService`. Le deuxième cas est de la forme :

```
Dim outilService As Object, unService As Object
Dim args(1) ' un ou plusieurs arguments
outilService = GetProcessServiceManager
' - initialiser le tableau args() avant cette instruction -
unService = outilService.createInstanceWithArguments( _
                  "com.sun.star.xxx.yyy.zzz", args())
```

Il existe aussi une troisième forme d'initialisation :

```
createInstanceWithArgumentsAndContext()
```

Ces deux dernières formes ne sont utilisées que dans des cas assez particuliers.

Avec COM Automation, le Service Manager est le premier objet à obtenir. Cet exemple en VB.NET obtient le Service Manager, puis l'objet correspondant à `StarDesktop`. Tous les autres services en découlent.

```
Public OpenOffice As Object, StarDesktop As Object
OpenOffice = CreateObject("com.sun.star.ServiceManager")
StarDesktop = OpenOffice.createInstance("com.sun.star.frame.Desktop")
```

CreateUnoService

Cette fonction permet d'obtenir un objet capable de fournir le service donné en argument :

```
Dim demandePasse As Object
demandePasse = CreateUnoService("com.sun.star.task.InteractionHandler")
```

Cette fonction est un raccourci pour :

```
Dim sm As Object, demandePasse As Object
sm = GetProcessServiceManager
demandePasse =
sm.createInstance("com.sun.star.task.InteractionHandler")
```

StarDesktop

Il s'agit d'un objet prédéfini qui est le service de base d'OpenOffice.org. Il est un raccourci pour :

```
Dim monOOo As Object
monOOo = CreateUnoService("com.sun.star.frame.Desktop")
```

ThisComponent

Représente l'objet document en cours ; il est en général équivalent à :

```
Dim monDocument As Object
monDocument = StarDesktop.getCurrentComponent()
```

Cependant, si l'EDI est en premier plan, `CurrentComponent` renverra l'EDI et la macro ne fonctionnera pas, alors que `ThisComponent` continue à renvoyer l'objet document. C'est pourquoi `ThisComponent` est préférable.

Notez que `ThisComponent` n'est pas une variable, mais un appel de fonction : s'il y a plusieurs documents OpenOffice.org ouverts, elle renvoie le document OpenOffice.org dont la fenêtre est actuellement en avant-plan. C'est pourquoi il est préférable de ne l'appeler qu'au début de la macro et de sauver le résultat dans une variable interne.

GetDefaultContext

Cette instruction est essentiellement utilisée pour accéder à un *singleton*. Elle est équivalente à :

```
GetProcessServiceManager.DefaultContext
```

`DefaultContext` est une propriété de l'objet ServiceManager.

Un singleton est un objet API qui ne peut exister qu'en un seul exemplaire. Voici un exemple d'utilisation :

```
dim sv As Object, repExt As String
sv = GetDefaultContext.getByName
 ➥ ("/singletons/com.sun.star.deployment.PackageInformationProvider")
repExt = sv.getPackageLocation("org.toto.test3")
```

CreateUnoStruct

Cette fonction sert à obtenir une structure UNO à l'exécution du programme, par exemple :

```
Dim uneProp As Object
uneProp = CreateUnoStruct("com.sun.star.beans.Property")
```

Habituellement on se contente de déclarer directement la variable :

```
Dim uneProp As New com.sun.star.beans.Property
```

CreateObject

Cette fonction a des usages multiples. Elle peut créer une structure UNO, comme `CreateUnoStruct`. Elle peut aussi créer d'autres objets OpenOffice ; le seul cas actuellement connu est un objet `Collection`, apparu pour une meilleure compatibilité avec VBA.

```
Dim coll As Variant ' ne pas utiliser le type Object !
coll = CreateObject("Collection")
```

La déclaration directe est aussi acceptée :

```
Dim coll As New Collection
```

La fonction `CreateObject` est souvent utilisée pour établir une connexion COM avec un objet externe (voir le chapitre 14).

```
Dim monWord As Object
monWord = CreateObject("Word.Application")
' est équivalent à :
Dim serviceCOM As Object, monWord As Object
serviceCOM = createUnoService("com.sun.star.bridge.oleautomation.Factory")
monWord = serviceCOM.createInstance("Word.Application")
```

Ici aussi, la déclaration directe est acceptée :

```
Dim monWord As New Word.Application
```

CreateUnoValue

Cette fonction sert à transmettre à l'API une donnée quelconque dans un type supporté par UNO. On l'emploie pour des problèmes très particuliers de conversion de

type de données vers l'API. Ici, on utilise une variable Basic d'un type `Long` et on la transmet sous forme d'un type `Hyper`, inexistant en Basic :

```
oFile.seek(CreateUnoValue("hyper", hxb)
```

Le tableau A-1 donne les équivalences entre type UNO et type Basic. Respectez la casse pour déclarer le type UNO.

Tableau A–1 Équivalence de types UNO et Basic

Type UNO	Type Basic
`boolean`	`Boolean`
`byte`	Non supporté (utiliser un `Integer` de -128 à +127)
`short`	`Integer`
`long`	`Long`
`hyper`	Non supporté (entier à 64 bits). Compatible dans la gamme des valeurs de `Long`.
`float`	`Single`
`double`	`Double`
`char`	Non supporté (utiliser la valeur Unicode en `Integer`)
`string`	`String` (limité à 65535 caractères)
`any`	`Variant`

L'équivalent de `CreateUnoValue` pour une programmation COM consiste à passer par une variable intermédiaire obtenue avec la méthode `Bridge_GetValueObject()` du Service Manager. Voir le chapitre *Automation Bridge* du *Developer's Guide*.

CreateUnoListener

Permet à un programme de s'enregistrer comme auditeur d'un ensemble d'événements. Nous en avons décrit le principe au chapitre 14.

CreateUnoDialog

Cette fonction sert à créer un dialogue (voir chapitre 11). Les langages non Basic peuvent aussi utiliser des dialogues OpenOffice, mais au prix d'instructions plus complexes. Reportez-vous au *Developer's Guide*, chapitre *Graphical User Interfaces, section Accessing Dialogs*.

IsUnoStruct

Cette fonction renvoie `True` si la variable est une structure UNO. Citons, comme exemple de structure UNO, le descripteur renvoyé par `createReplaceDescriptor` (voir chapitre 8). Cette fonction permet de distinguer une structure d'un objet véritable, ou d'une donnée simple.

Les variables de structure UNO sont de vraies valeurs, et non des références comme les variables sur les objets. C'est la raison des recopies nécessaires pour effectuer une modification, comme par exemple ici :

```
dim rognure as object
rognure = monImage.GraphicCrop
rognure.Bottom = -2000
monImage.GraphicCrop = rognure
```

EqualUnoObjects

Les variables représentant des objets sont en fait des références sur l'objet lui-même. Il est donc possible d'avoir deux variables pointant sur le même objet. Cette fonction permet de le vérifier.

HasUnoInterfaces

Cette fonction renvoie `True` si l'objet en premier argument prend en charge toutes les interfaces listées en arguments.

```
if HasUnoInterfaces(monObjet, "com.sun.star.embed.XVisualObject", _
   "com.sun.star.frame.XStorable2") then
```

Cette fonction est parfois utile pour rechercher un objet d'un type particulier dans une collection hétéroclite. Un autre moyen est d'utiliser la fonction `supportsService` d'un objet pour vérifier s'il offre le service souhaité.

ConvertToURL, ConvertFromURL

Ces deux fonctions Basic font envie aux programmeurs utilisant d'autres langages. En effet, convertir une adresse Windows en URL n'a rien d'évident si l'adresse comporte des espaces ou des caractères nationaux. Pour des adresses de fichiers, ces deux fonctions sont équivalentes au codage suivant, qui utilise le service API `FileContentProvider` :

```
Dim sv As Object
Dim adr1 As String, adr2 As String, adrURL As String
```

```
adr1 = InputBox("Donnez une adresse système")

sv = CreateUnoService("com.sun.star.ucb.FileContentProvider")
adrURL = sv.getFileURLFromSystemPath("", adr1) ' ConvertToURL
adr2   = sv.getSystemPathFromFileURL(adrURL)   ' ConvertFromURL
' adr1 doit être identique à adr2
MsgBox("Adresse système 1 : " & adr1 & chr(13) & _
       "Adresse système 2 : " & adr2 & chr(13) & _
       "Adresse URL : " & adrURL)
```

Un script Python importera le module `uno` afin d'utiliser les fonctions `systemPathToFileUrl` et `fileUrlToSystemPath` qui sont l'équivalent de `ConvertToURL` et `ConvertFromURL`, respectivement.

La documentation de l'API et le Software Development Kit

L'ensemble de la documentation est appelé SDK (*Software Development Kit*), conçu à l'origine pour un environnement de développement en Java ou en C++. Depuis la version 3 d'OpenOffice.org, son contenu téléchargeable a été réduit en éliminant le *Developer's Guide*.

Le *Developer's Guide* est un hypertexte expliquant la conception de l'API OpenOffice.org. Il est disponible dans le wiki d'OpenOffice.org à l'adresse :

http://wiki.services.openoffice.org/wiki/Documentation/DevGuide/OpenOffice.org_Developers_Guide

Ce wiki permet de créer un fichier PDF de certaines parties. Mais le *Developer's Guide* contient de nombreux liens vers la référence API disponible aussi en ligne, et réciproquement, ce qui rend une lecture interactive souvent préférable à une lecture imprimée.

Le SDK est téléchargeable à l'adresse http://api.openoffice.org/SDK/. C'est un exécutable qui installe sur votre ordinateur :

- la référence IDL (*Interface Definition Language*) qui est un gigantesque hypertexte documentant (presque) tous les objets (au sens le plus général) de l'API ;
- les fichiers `*.idl` ayant servi à constituer la référence ; ce sont des fichiers texte où on peut lire les valeurs des constantes nommées ;
- une référence pour le développement en Java ;
- une référence pour le développement en C++ ;
- des exemples (dont ceux employés dans le *Developer's Guide*) ;
- la spécification des formats XML utilisés ;

- divers outils de développement.

Lorsqu'on étudie cette documentation, la principale difficulté que l'on rencontre est qu'elle est conçue pour un développeur Java ou C++ chargé de faire évoluer OpenOffice.org ou d'en réaliser une variante. Ce point de vue est intimement mélangé avec les descriptions des fonctionnalités disponibles, ce qui en rend la lecture assez difficile. De plus, et principalement dans le *Developer's Guide*, les exemples sont donnés en langage Java, notablement plus lourd que OOoBasic.

La référence IDL est rédigée par les programmeurs eux-mêmes lors du développement, et compilée ensuite automatiquement. L'ennui, c'est que les programmeurs sont rarement intéressés par l'écriture de la documentation. Aussi est-elle parfois décevante par son aspect répétitif et ses lacunes.

Comment s'y retrouver ?

Nous vous conseillons d'installer le SDK sur votre ordinateur, pour un accès plus rapide, et d'utiliser un navigateur Internet capable d'afficher de multiples pages accessibles sous forme d'onglets.

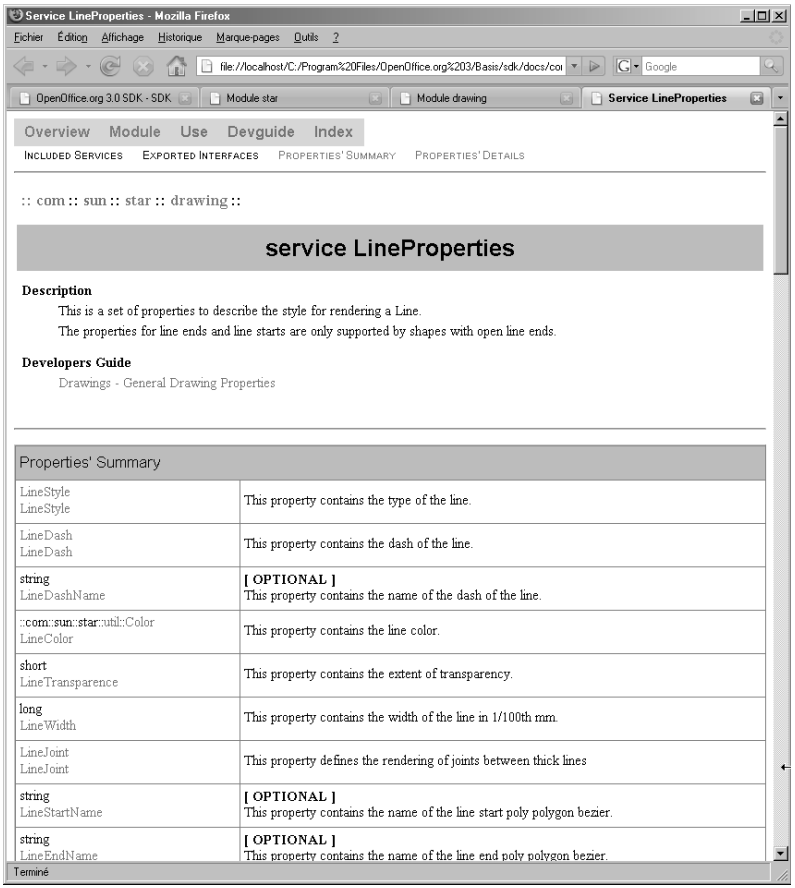
Dans une installation standard sous MS-Windows, et avec la version 3 d'OpenOffice.org, l'ensemble du SDK est installé dans le répertoire `C:\Program Files\OpenOffice.org 3\Basis\sdk`. Dans ce répertoire, affichez dans un navigateur le fichier `index.html`. La page affichée comporte un lien vers le wiki du *Developer's Guide* et un lien vers *IDL Reference*.

La première page de la référence IDL liste les différents modules qui composent le module `Star`, l'ensemble de l'application. Tout est ensuite décomposé en un arbre avec de nombreuses ramifications. La position d'une page dans cet arbre est rappelée en haut à gauche des pages (par exemple `::com::sun::star::`) et l'organisation des répertoires contenant les pages HTML reflète exactement cette hiérarchie. Lorsque nous indiquons une référence de documentation comme `com.sun.star.drawing.LineProperties`, vous afficherez la page correspondante en suivant, depuis la première page de l'IDL, le lien intitulé `drawing`, puis dans la page obtenue le lien intitulé `LineProperties`, comme on peut le voir sur la figure A-1.

L'autre moyen d'accès à l'IDL, à partir de n'importe laquelle de ses pages, est d'utiliser le lien Index, sur la première ligne du haut de la page. Il vous affiche la page `A` d'un dictionnaire. Si vous cherchez la documentation sur `LineProperties`, affichez la page `L` et recherchez ce mot dans celle-ci, en début de ligne. Vous verrez la ligne :

```
LineProperties - service ::com::sun::star::drawing:: .LineProperties
```

Figure A–1
Une page de la référence IDL



Ici, il s'agit d'un service dont la page de description est accessible par lien hypertexte. Dans bien des cas, vous obtiendrez plusieurs lignes avec le même nom, car plusieurs types d'objets ayant une propriété ou une méthode similaire portent normalement le même nom. Ce sera à vous de déterminer lequel correspond à votre contexte, grâce aux autres informations de la ligne.

Un dernier moyen, très rapide, d'accéder directement à la bonne page de l'IDL consiste à utiliser l'outil XRay.

Xray

L'API offre des fonctions dites d'introspection et de *core reflection* permettant d'obtenir à l'exécution de nombreuses informations sur les objets manipulés. Cependant, ces fonctions sont assez complexes à utiliser.

L'outil Xray, réalisé avec des macros, met en forme ces informations et permet d'étudier les sous-objets. Il est capable de retrouver dans l'IDL la documentation du sous-objet, si elle existe. Les auteurs utilisent intensivement Xray pour étudier la structure des objets et lire leur documentation dans l'API. Sans Xray, cet ouvrage ne serait pas aussi détaillé.

Xray est un logiciel Open Source disponible en téléchargement :

- en version française depuis la page http://fr.openoffice.org/Documentation/How-to/indexht-programmation.html, section programmation Basic, cinquième document,
- en version anglaise sur OOoMacros http://ooomacros.org/dev.php#101416.

Xray se présente sous la forme d'un document explicatif écrit au format `sxw` car il est entièrement compatible avec les anciennes versions d'OpenOffice.org. En cliquant un bouton du document, les bibliothèques de macros de Xray sont installées dans Mes Macros.

Pour étudier un objet il est nécessaire d'insérer une instruction dans le codage, pour appeler Xray en donnant cet objet en argument :

```
monDocument = thisComponent
lesFeuilles = monDocument.Sheets
maFeuille = lesFeuilles.getByName("Janvier")
xray maFeuille
```

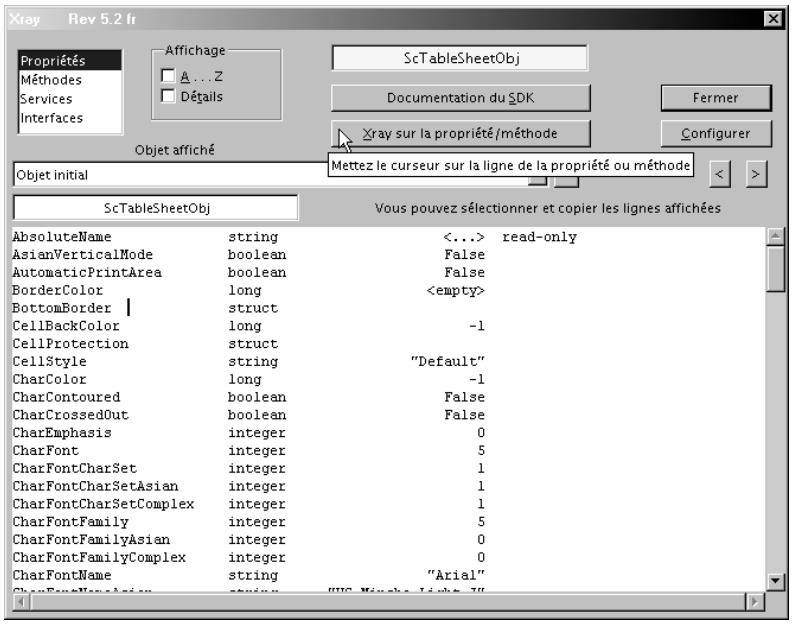
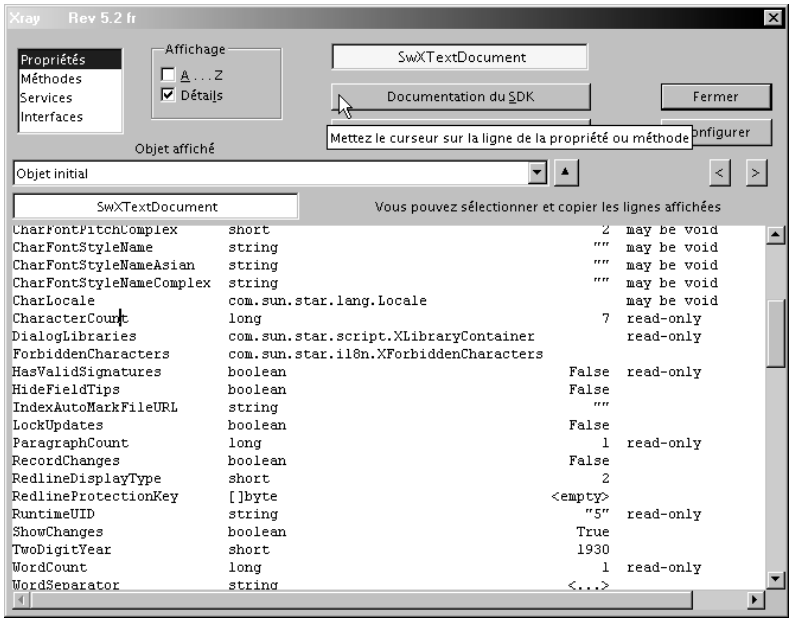


Figure A–2
Xray : propriétés d'une feuille Calc

On obtient le panneau de la figure A-2, où on visualise les propriétés disponibles pour l'objet analysé. En positionnant le curseur sur la ligne d'un élément affiché, il suffit de cliquer sur le bouton Xray pour afficher le contenu de ce sous-objet. L'opération peut être répétée sur différents sous-objets, à plusieurs niveaux.

En positionnant le curseur sur la ligne d'un élément, un simple clic sur le bouton Documentation permet de visualiser sur votre navigateur Internet la documentation API concernant l'objet sélectionné (voir la figure A-3). Pour utiliser cette fonction, il faut toutefois avoir installé la documentation SDK sur l'ordinateur.

Figure A–3
Xray : voir la documentation API



La figure A-4 montre les méthodes offertes par l'objet en analyse. Ici l'affichage est en mode détaillé, et en ordre alphabétique. On peut approfondir l'analyse sur une des méthodes à condition qu'elle ne comporte aucun argument ou encore accéder à sa documentation.

Xray vous permet de lister les services proposés par l'objet et les services disponibles par invocation, ou lister les interfaces prises en charge. Là encore, la documentation d'un service ou d'une interface est directement accessible.

Il faut une certaine expérience de l'API pour ne pas être perdu dans les informations fournies. Comme les objets API héritent pour la plupart des propriétés et méthodes d'un ou plusieurs autres objets, et ainsi de suite, et que vous pouvez analyser un objet interne à un objet, vous retrouvez certaines méthodes et propriétés des « briques de

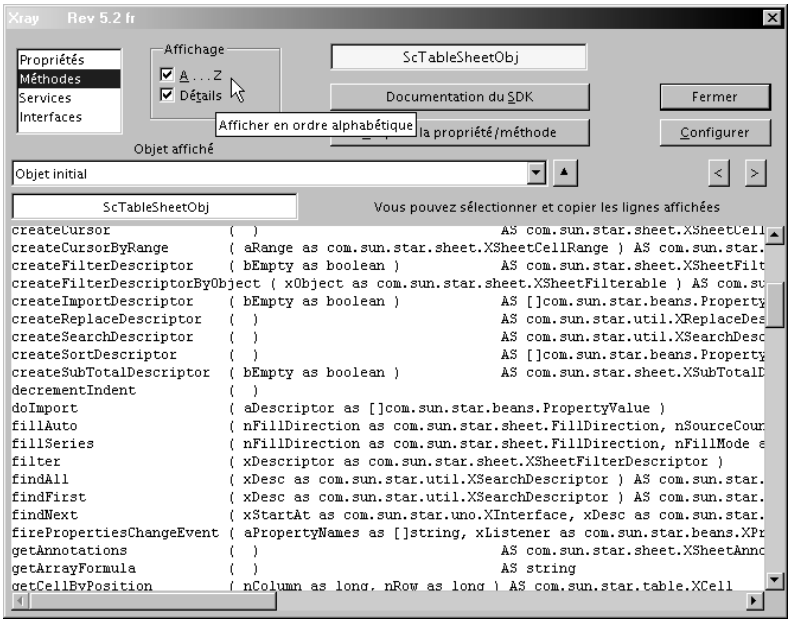


Figure A–4
Xray : méthodes d'une feuille Calc

base » d'OpenOffice.org, par exemple `setDelegator`, `getByName`, ou le tableau `ImplementationId`. Certaines propriétés d'objet sont listées mais non disponibles dans le contexte, d'autres renvoient un objet qui n'est autre que l'objet lui-même.

Lorsque vous demandez la documentation à propos d'un élément d'un objet, il arrive que la recherche échoue : la documentation manque, ou l'élément est un « fossile » des temps révolus, ou l'objet n'est pas officiellement utilisable.

Object Inspector

Cet outil permet lui aussi d'explorer un objet API. Il est décrit en anglais à la page http://wiki.services.openoffice.org/wiki/Object_Inspector. C'est une extension écrite en Java. Bien que plus complexe que Xray, il est intéressant pour des développements en Java ou C++ grâce à sa faculté de produire des portions de code.

MRI

Écrit par le japonais Hanya, cet outil est calqué sur Xray. Il est développé en Python sous la forme d'une extension qui introduit un nouveau service. Il est capable de modifier des valeurs de propriétés et d'exécuter des méthodes avec arguments, ce qui peut être utile avec des langages qui n'ont pas la souplesse de l'EDI Basic. La documentation est en anglais et japonais.

Vous trouverez MRI à la page http://extensions.services.openoffice.org/project/MRI.

Conclusion

Cette annexe vous a apporté les connaissances minimales pour étudier l'API. Grâce à Xray et au SDK le développeur pourra progressivement maîtriser les nombreuses possibilités qu'offre l'API.

Le chapitre suivant offre un panorama de routines utilitaires, qui complète utilement les techniques vues au chapitre 14.