OpenOffice::OODoc

The Perl Open OpenDocument Connector

Guide de prise en main

Une interface de programmation Perl / OpenDocument

L'original de ce document est accessible suivant le lien
http://jean.marie.gouarne.online.fr/doc/oodoc_guide.odt

Version 1.7

Août 2006

Sommaire

1 Introduction

Le module OpenOffice::OODoc est une interface de programmation destinée au traitement de documents au format *OASIS OpenDocument* (ISO/IEC 26300) ou au format « historique » d'*OpenOffice.org*. Cette interface, basée sur Perl, est orientée objet. Elle est complètement indépendante de l'environnement de développement OpenOffice.org et donne directement accès aux fichiers, sans utiliser de logiciel bureautique. Elle permet de créer, de consulter, de modifier ou de supprimer tout élément de contenu ou de présentation dans tout document.

L'objectif de ce module est de faciliter l'utilisation du format OpenDocument dans le cadre de traitements automatiques tels que :

- génération ou personnalisation de documents à partir de chaînes de traitement ou de bases de données extérieures à l'environnement bureautique (rapports, tableaux de bord, publipostages) ;
- recherche et exploitation de données saisies dans des documents bureautiques (traitement de formulaires, gestion documentaire) ;
- mutualisation et réutilisation d'éléments de contenu ou de présentation (styles, formules légales ou commerciales prédéfinies, logos, etc.) à l'aide de services distribués (SOA, web services) et/ou de bases de données (XML ou généralistes).

Les domaines d'application sont dont très divers. Le logiciel étant disponible en téléchargement anonyme, il n'existe pas de statistiques d'utilisation mais, d'après les échanges que j'ai parfois avec des développeurs de plusieurs continents, il semble avoir été mis en oeuvre dans des secteurs variés comme la santé, les professions juridiques, l'industrie alimentaire, l'exploration pétrolière, l'électronique, l'enseignement, l'immobilier, etc.

Le format OpenDocument, fort des quelques années de mise en oeuvre opérationnelle de son prédécesseur OpenOffice.org[1] et de son statut de standard international[2], semble avoir le vent en poupe. Sans entrer dans des considérations technologiques, économiques et politiques qui nous écarteraient du sujet de ce guide, on peut dire que l'arrivée d'OpenDocument est une occasion unique pour l'utilisateur, quel que soit son métier, de se réapproprier ses documents. C'est-à-dire notamment de les exploiter comme bon lui semble, sans être obligé d'en passer par les interfaces de programmation

1 Il faut évidemment distinguer le *format* OpenOffice.org du *logiciel bureautique* OpenOffice.org, même si c'est le second qui a permis au premier d'exister et d'être reconnu.

2 Le format OpenDocument a été créé par l'OASIS (http://www.oasis-open.org) sous la dénomination "*OASIS Open Document Format for Office Applications*" et adopté par l'ISO (http://www.iso.org) sous la référence *ISO/IEC 26300*.

propres à tel ou tel logiciel bureautique. Comme je le répète avec assiduité, *le document doit avoir une vie indépendante du logiciel qui l'a créé*. OpenOffice::OODoc est l'un des outils destinés à mettre ce principe en application.

Ce guide est une aide au démarrage. Il ne doit surtout pas être considéré comme un manuel de référence car il ne présente qu'une petite partie de l'interface de programmation. Son objectif est de permettre à l'utilisateur de comprendre facilement la logique de cette interface, et de développer très rapidement des programmes simples mais efficaces. Dans ce but, la liste des fonctionnalités décrites est sciemment restreinte par rapport au manuel complet ; certaines possibilités sont délibérément laissées de côté, et la présentation est organisée par thèmes et non par modules techniques.

Le véritable manuel de référence de l'interface de programmation OpenOffice::OODoc est fourni (en Anglais) avec la distribution standard. Il est bien sûr possible, sans rien installer, de le consulter en ligne sur le site du projet[3]. Ce manuel est présenté selon un découpage par modules. Il est à peu près complet mais, en contrepartie, son maniement est plutôt rébarbatif quand on n'a pas déjà une idée générale des caractéristiques et du fonctionnement de l'outil. D'où l'intérêt (j'espère) de ce guide introductif, dont la compréhension suppose simplement la connaissance de Perl et des principes élémentaires de la programmation orientée objet (la connaissance de XML peut faciliter les choses, mais elle n'est pas indispensable).

Il est bien entendu recommandé, en plus, de connaître les principales fonctions d'OpenOffice.org, au moins en tant qu'utilisateur, ne serait-ce que pour pouvoir comprendre et contrôler les documents utilisés et produits par les programmes.

3 http://search.cpan.org/dist/OpenOffice-OODoc

2 Installation

Vous pouvez sauter ce chapitre si vous êtes sûr que la boîte à outils OpenOffice::OODoc est déjà correctement installée.

Il existe bien entendu un grand nombre de façons de faire cette installation. Nous en resterons ici aux plus classiques et aux plus sûres.

2.1 Pré-requis d'environnement

OpenOffice::OODoc a été testé sur un grand nombre de plates-formes, parmi lesquelles Windows, Linux, Solaris, HP-UX, FreeBSD, Darwin, etc. Il fonctionne théoriquement sur toute plate-forme supportant Perl.

La version 5.8.0 de Perl (ou une version plus récente) est requise.

Les modules Perl Archive::Zip et XML::Twig sont également requis, ainsi que d'autres modules qui, eux, sont presque toujours fournis en standard avec Perl (File::Basename, File::Temp, File::Copy, Time::Local, Encode).

OpenOffice::OODoc est en « pur Perl »; son installation ne nécessite aucune compilation et son code est exactement le même pour toutes les plates-formes. Toutefois, l'installation des modules Archive::Zip et XML::Twig dont il dépend peut entraîner d'autres dépendances qui, elles, requièrent la présence d'un compilateur C.

Remarque importante : OpenOffice::OODoc est une interface de programmation entièrement indépendante de celle du logiciel bureautique OpenOffice.org. Pour la vérification des documents créés ou modifiés par les programmes qui l'utilisent, il est préférable d'avoir ce logiciel bureautique sous la main. Mais, à l'exécution, le programmes développés avec cette boîte à outils peuvent s'exécuter sans qu'OpenOffice.org soit installé. Ils peuvent d'ailleurs être déployés sur des plates-formes pour lesquelles OpenOffice.org n'est pas disponible.

2.2 Installation manuelle à partir de la distribution CPAN

Si toutes les dépendances ci-dessus sont satisfaites, il est possible de faire simplement une installation manuelle à partir de l'archive tar.gz disponible sur le site CPAN. Cette archive est téléchargeable via le lien [Download] dans la page

http://search.cpan.org/dist/OpenOffice-OODoc

Le fichier se nomme OpenOffice-OODoc-x.xxx.tar.gz (où x.xxx dépend du numéro de version).

Les étapes de l'installation sont les suivantes :

1. décompresser l'archive dans un répertoire de travail, ce qui aura pour effet de créer un sous-répertoire OpenOffice-OODoc-x.xxx ;
2. entrer dans ce sous-répertoire et exécuter la commande « perl Makefile.PL » ; cette commande va provoquer une question, qu'il est d'ailleurs possible d'ignorer dans un premier temps en tapant simplement « Entrée » ; nous y revenons plus loin ;
3. exécuter ensuite « make test » et vérifier l'absence de message d'erreur ;
4. si tout va bien, exécuter « make install » pour installer effectivement le module, sa documentation et les exemples.

Sous Windows, l'utilitaire MAKE n'est pas forcément disponible. Il peut être remplacé par NMAKE dans les commandes ci-dessus. Dans ce cas, il sera peut-être nécessaire d'avoir installé au préalable le programme NMAKE.EXE (disponible en téléchargement sur le site de Microsoft[4]).

Il est à noter que, en cas de succès, la commande « `make test` » (ou « `nmake test` ») provoque la création d'un document OpenOffice.org Writer dans le répertoire courant de l'installation. Ce document, qui contient un compte-rendu sommaire de l'installation, peut être ouvert sous OpenOffice.org pour vérification.

Une fois l'installation réussie, le répertoire dans laquelle elle a été faite peut être supprimé.

Parlons un peu des options d'installation que l'utilisateur est invité à valider par un message (en Anglais) qui apparaît en réponse à la commande « `perl Makefile.PL` ». Elles sont actuellement (en version 2.x) au nombre de quatre :

1. Le jeu de caractères de l'installation. Il est possible de le laisser à sa valeur par défaut (iso-8859-1) si on ne compte pas créer, modifier ou rechercher des contenus incluant des symboles exclus de cet alphabet, comme le signe « € » (même si les documents en contiennent par ailleurs). De plus, le choix du jeu de caractères tel qu'il est fait à ce stade ne lie pas définitivement les programmes, puisqu'il existe une fonction `ooLocalEncoding` qui permet de changer de jeu de caractères à tout moment. Cependant, en Europe Occidentale, il est recommandé de choisir par exemple cp-1252 (sous Windows) ou iso-8859-15 (sous Unix). Voir le module Perl Encode pour connaître la liste des jeux de caractères supportés sur votre plate-forme. Le choix d'un jeu de caractères non supporté provoquera l'affichage d'une alerte mais n'empêchera pas l'installation.

4 http://download.microsoft.com/download/vc15/Patch/1.52/W95/EN-US/Nmake15.exe

2. La « table des couleurs » (*colormap*). C'est sans doute, pour certains, un gadget inutile et, par défaut, il n'y en a pas. Cette table, si elle est installée, fournit des noms de couleurs « en clair » pour les programmes qui consultent ou modifient les attributs de couleurs dans les styles de présentation (ex: couleurs de fond, couleurs de texte, etc). À défaut, le programmeur doit fournir le triplet numérique RVB (rouge, vert, bleu) définissant chaque couleur de son choix. Certains noms de couleurs de base sont codés « en dur » dans OpenOffice::OODoc en l'absence de table de couleurs ; ainsi, on pourra quand même écrire « `pink` » au lieu de (`255, 192, 203`) ou « `yellow` » au lieu de (`255, 255, 0`). Mais il existe des fichiers de correspondance au format RGB.TXT, librement téléchargeables, ou tout simplement créés par l'utilisateur, qui définissent des dizaines ou des centaines de noms symboliques de couleurs. Le fichier RGB.TXT standard de l'environnement *X-Window* ou *Xorg* peut être utilisé tel quel (y compris sous Windows). Dans un environnement Unix/Linux, il est souvent présent sous

   ```
   /usr/lib/X11/rgb.txt
   ```

3. Le répertoire de travail pour les fichiers temporaires. Il faut savoir que, quand OpenOffice::OODoc crée ou modifie un fichier ODF, il produit des fichiers temporaires (automatiquement supprimés après emploi, sauf incident). Par défaut, ces fichiers sont placés dans le répertoire courant de l'application en cours. Cette option permet de définir un répertoire de travail commun et fixe.

4. Le format documentaire à utiliser par défaut quand un programme crée de toutes pièces un nouveau document. L'option 1 signifie que les documents générés seront au format OpenOffice.org version 1. L'option 2 (par défaut) indique qu'ils seront plutôt au format *OASIS Open Document*, alias ODF, alias « OpenOffice 2 ». Attention, depuis la version 2.021, le format 2 est le format par défaut, ce qui n'était pas le cas dans les versions précédentes. Par la suite, il est probable que l'ODF remplacera complètement l'ancien format et que vous n'entendrez plus parler de cette option. Deux détails sont cependant à retenir ici. D'abord, tout programme peut choisir à tout moment une autre option que la valeur choisie ici. Ensuite, cette option ne vaut que pour les document *entièrement créés* par les programmes utilisant OpenOffice::OODoc ; elle est sans effet pour les document préexistants (dont le format est reconnu automatiquement).

Si toutes ces options vous ennuient, vous pouvez remplacer la commande

```
perl Makefile.PL
```

par

```
perl Makefile.PL –noconfig
```

et dans ce cas aucune question ne vous sera posée (la configuration par défaut sera appliquée).

2.3 Installation automatique CPAN

Si le module CPAN est présent et configuré dans votre environnement Perl, et bien sûr si vous avez un accès internet, le téléchargement et l'installation peuvent être réalisés automatiquement par la ligne de commande suivante :

```
perl -MCPAN -e 'install OpenOffice::OODoc'
```

Au cours de l'exécution de cette commande, il faudra quand même choisir les paramètres de configuration décrits à propos de la procédure manuelle (ou taper « Entrée » si les valeurs par défaut conviennent). Le test sera ensuite exécuté automatiquement et, en cas de succès, sera suivi par l'installation proprement dite.

Le principal avantage de cette procédure est la vérification des dépendances et l'installation automatique des pré-requis s'il ne sont pas déjà présents.

2.4 Installation automatique PPM

OpenOffice::OODoc, bien que sa distribution de référence soit celle du CPAN, est également disponible en distribution PPM (*Perl Package Management*). Les utilisateurs équipés de l'environnement ActivePerl 5.8.x peuvent l'installer via la commande :

```
ppm install OpenOffice::OODoc
```

Comme l'installation CPAN, l'installation PPM assure le respect des dépendances. Elle présente cependant, du moins jusqu'à présent, quelques limitations. Par exemple :

- le test avant installation n'est pas exécuté (cet inconvénient est relatif sachant qu'il existe une distribution PPM distincte pour chaque plate-forme, et que cette distribution n'est diffusée que si elle a été testée pour la plate-forme concernée) ;
- l'utilisateur n'est pas nécessairement invité à choisir les options d'installation indiquées ci-dessus ; pour personnaliser ces options, il doit éditer le fichier `config.xml` après l'installation (voir plus loin) ;
- la distribution PPM n'est pas nécessairement à jour par rapport à la distribution CPAN[5].

5 On peut notamment vérifier sur les sites PPM d'*ActiveState* (http://ppm.activestate.com) et de l'*Université de Winnipeg* (http://theoryx5.uwinnipeg.ca/ppms) si, pour chaque plate-forme, la distribution PPM est au niveau de la distribution CPAN.

2.5 Autres types d'installation

Le nombre de distributions OpenOffice::OODoc tend à augmenter, et il n'est pas possible d'explorer ici tous les cas de figure.

La dernière née est la distribution RPM[6] officielle pour *Mandriva Linux 2007*.

Je maintiens moi-même une distribution RPM *Mandriva* non officielle, à jour et compatible avec des environnements *Mandrake*/*Mandriva* plus anciens, dans le dossier suivant :

http://jean.marie.gouarne.online.fr/tech/oodoc

D'autres (que je ne connais pas toutes) sont apparues de ci, de là. Chacun peut faire son marché et choisir selon ses préférences...

2.6 Le fichier de configuration

Les options de configuration que l'utilisateur est invité à accepter ou à modifier lors de l'installation de référence (CPAN) sont enregistrées dans un fichier XML très court. Ce fichier est lui-même accessible sous <installation>/OpenOffice/OODoc/config.xml (où <installation> est le chemin d'accès aux modules Perl, dépendant de votre environnement.

Voici à quoi ressemble ce fichier, dans la configuration par défaut :

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
      <comment>OpenOffice::OODoc local configuration file</comment>
      <OpenOffice-OODoc>
            <XPath-LOCAL_CHARSET>iso-8859-1</XPath-LOCAL_CHARSET>
            <File-WORKING_DIRECTORY>.</File-WORKING_DIRECTORY>
            <Styles-COLORMAP></Styles-COLORMAP>
            <File-DEFAULT_OFFICE_FORMAT>2</File-DEFAULT_OFFICE_FORMAT>
            <INSTALLATION_DATE>2006-01-21T21:59:24</INSTALLATION_DATE>
      </OpenOffice-OODoc>
</config>
```

Pour sélectionner un autre jeu de caractères, il suffit de remplacer « iso-8859-1 » par la valeur appropriée dans l'élément `<Xpath-LOCAL_CHARSET>`. De même, pour le répertoire de travail, le point « . » (répertoire courant) peut être remplacé par le chemin complet d'un répertoire quelconque dans l'élément `<File-WORKING_DIRECTORY>`. Et pour choisir un fichier de couleurs, il suffit d'insérer son chemin d'accès complet entre les balises `<Styles-COLORMAP>` (non renseigné par défaut).

6 *Red Hat Package Management*, format de distribution et d'installation créé par *Red Hat* et adopté par plusieurs distributions Linux.

2.7 Utilisation

Une fois le logiciel installé, il s'exploite comme n'importe quel autre module Perl, soit via une instruction `use` ou `require` :

```
use OpenOffice::OODoc;
```

Après une clause `use`, les programmes peuvent appeler des fonctions d'initialisation s'il est nécessaire de forcer des valeurs de configuration particulières (différentes de celles du fichier `config.xml`). La séquence suivante, par exemple, utilise, pour le programme courant, un jeu de caractères, un fichier de couleurs et un répertoire de travail spécifiques :

```
use OpenOffice::OODoc;
ooLocalEncoding "cp-1252";
ooLoadColorMap "C:\Documents\couleurs.txt";
ooWorkingDirectory "C:\Windows\Temp";
```

3 Les fichiers OpenDocument

L'interface de programmation OpenOffice::OODoc est conçue pour masquer les détails techniques du stockage des documents, mais il est quand même utile d'en connaître l'organisation générale pour comprendre rapidement comment vont fonctionner les applications. Attention, ce chapitre se borne à présenter le peu que l'utilisateur à intérêt à connaître pour utiliser l'outil ; ce n'est pas la spécification du format OpenDocument !

3.1 Archive et Membres

Techniquement, un fichier OpenDocument est une archive compressée de type « zip », contenant elle-même plusieurs fichiers dont chacun a une fonction particulière. Je ne m'attarde pas ici à l'énumération de ces fichiers. En me plaçant à un niveau logique plutôt que physique, je dirai simplement que chacun des ces fichiers correspond à ce que nous appelons un « membre »[7] dans la suite de ce guide (comme dans le manuel de référence). Le nombre de membres n'est pas limité a priori, mais certains d'entre eux sont toujours présents car ils contiennent la combinaison minimale d'informations nécessaire pour la reconstitution d'un document complet. Parmi les membres obligatoires, certains sont plus souvent sollicités que d'autres, et j'en donne donc immédiatement la liste :

1. « ***content*** », c'est-à-dire en Français « contenu », représente le contenu textuel du document. Ce contenu est constitué par exemple de paragraphes, de titres, de listes, de tableaux, de notes, et autres objets contenant du texte. Détail important, « content » ne contient pas, par exemple, les images que l'utilisateur a pu insérer dans les documents, même si, en apparence, ces images semblent intégrées au contenu. En réalité, les images incrustées dans une feuille Calc, une page Writer, ou une diapositive Impress sont visuellement présentées à leur place logique par OpenOffice.org, mais elles résident chacune dans un « membre » distinct. « content » contient éventuellement des liens sur des images, mais pas les images elles-mêmes.
2. « ***styles*** » représente la définition de tous les styles *nommés* (créés par l'utilisateur ou prédéfinis), dont la liste apparaît dans le « styliste » d'OpenOffice.org (activé par la touche F11). Ces styles sont tous réutilisables. Attention, il existe aussi des styles « automatiques », qui n'ont pas de nom et qui n'apparaissent pas dans la liste. Par exemple, si vous sélectionnez un mot dans une phrase dans OpenOffice.org et si vous le mettez, disons, en italiques et en jaune sur fond noir, vous créez en réalité un style spécial pour ce mot. Ce style est enregistré dans le membre « `content` », et

7 Dans le vocabulaire du module Perl Archive::Zip (utilisé par OpenOffice::OODoc), un membre (*member*) est une unité de contenu intégrée dans une archive.

non pas dans « `styles` », parce qu'il n'est pas destiné à être réutilisé et parce qu'il est exclusivement associé à un contenu et un seul.

3. « ***meta*** » représente les « métadonnées », c'est-à-dire à peu de choses près les informations que l'utilisateur peut voir ou modifier dans la boîte à onglets activée via l'item « Propriétés » du menu « Fichier » dans OpenOffice.org. Ce sont les propriétés globales du document, telles que le titre, le sujet, les mots-clés, etc.

« `content` », « `styles` » et « `meta` » sont des membres XML, ce qui veut dire que chacun d'eux possède une structure arborescente et constitue, et peut être vue comme une base de données à laquelle, via OpenOffice::OODoc, les programmes se « connectent » et adressent des « requêtes ».

Pour la plupart des opérations relatives au contenu et à la présentation des documents, l'interface d'accès privilégiée est l'objet « `Document` », créé par la fonction d'initialisation `ooDocument()`, tandis que les accès aux métadonnées relèvent de l'objet « `Meta` » créé par la fonction `ooMeta()`. Nous verrons cela un peu plus loin.

3.2 Éléments

Les constituants fondamentaux de chacun de ces membres sont appelés des *éléments*, chaque élément étant un conteneur renfermant une portion de texte et/ou des attributs, selon sa fonction. Par exemple un élément « paragraphe » est un conteneur possédant d'une part un texte (le texte du paragraphe, tout simplement) et un attribut de style indiquant le nom du style à utiliser pour représenter ce paragraphe. Un titre (*heading*) est un autre type usuel d'élément ; il possède non seulement un texte et un style, comme un paragraphe, mais aussi un niveau hiérarchique.

Certains attributs sont des clés permettant d'établir des liens vers entre éléments. C'est notamment le cas de l'attribut de style d'un élément textuel (paragraphe, titre ou autre). La philosophie d'OpenDocument interdisant de mêler contenu et apparence, il n'est pas question qu'un élément textuel possède des attributs de présentation. L'attribut de style contient donc simplement le nom (identifiant) d'un style, qui est la clé d'accès à un élément style défini ailleurs.

Un style « automatique » peut être défini dans « `content` » et non dans « `styles` », mais toujours dans un élément distinct du conteneur de texte auquel il s'applique. Un élément « style », quant à lui, ne contient pas de texte[8] (il est fait pour être appliqué à des textes portés par des conteneurs de texte), mais possède des attributs de présentation qui peuvent être très nombreux (couleurs d'avant-plan et d'arrière-plan,

8 On peut nuancer cette règle à propos des styles de pages (présentés plus loin). Un style de page contient notamment, le cas échéant, des descripteurs d'entêtes et de pieds de pages, qui peuvent contenir eux-mêmes des éléments textuels.

taille des caractères, police, etc.). Un style peut lui-même être dérivé d'un autre style ; il possède dans ce cas un attribut « parent » qui n'est autre que le nom du style dont il est l'héritier.

Un grand nombre de méthodes d'accès sont prédéfinies pour un grand nombre d'éléments. La forme générale d'une méthode de recherche d'élément est `$doc->getXXX(condition)` ou `$doc->selectXXX(condition)`, `$doc` étant la référence d'un objet Document préalablement initialisé (voir plus loin). Exemples :

```
$p = $doc->getParagraph(10);
$e = $doc->selectElementByContent('(B|b)ureautique');
```

Ici, `$p` contient un paragraphe sélectionné d'après sa position, tandis que `$e` contient le premier élément (paragraphe ou autre) dont le texte contient « Bureautique » ou « bureautique », ces deux éléments étant pris dans l'espace du document `$doc`.

L'utilisateur avancé peut toujours, si nécessaire, avoir recours à des expressions XPath pour accéder à des éléments non pris en charge par des méthodes d'accès prédéfinies, en utilisant des méthodes comme `getElement()` ou `getNodeByXPath()`, décrites dans OpenOffice::OODoc::XPath. Cette approche nécessite cependant une certaine connaissance de la structure XML des documents et de la syntaxe XPath[9]. Elle ne doit être employée qu'en cas de nécessité, et je n'en dirai pas plus dans ce guide.

Une fois que l'application dispose d'un élément, sélectionné par l'une de ces méthodes, elle peut agir sur lui par des méthodes de lecture ou de modification de la forme `$doc->XXX($element, [arguments])`. Pour les utilisateurs avancés, connaissant le module XML::Twig[10], il est également possible d'exécuter des actions de la forme `$element->XXX`, sachant que les éléments dont il est question avec OpenOffice::OODoc sont des objets de classe XML::Twig::Elt, mais nous n'insistons pas ici sur ce point.

Exemple :

```
$style = $doc->textStyle($p);
$doc->textStyle($e, $style);
$contenu = $doc->getText($p);
$doc->extendText($e, $contenu);
```

La séquence ci-dessus, dont toutes les instructions agissent dans le document `$doc`, récupère le style du paragraphe `$p` et l'applique à l'élément `$e`, puis elle ajoute le texte de `$p` à la fin de `$e`.

9 Attention, OpenOffice::OODoc ne supporte pas l'intégralité du standard XPath. Par ailleurs, une expression XPath conçue pour un document OpenOffice.org 1.x n'est pas forcément applicable telle quelle à un document au format Open Document, et vice versa.

10 http://search.cpan.org/dist/XML-Twig

Après ces quelques indications, vous comprendrez mieux pourquoi, lors de la connexion à un document (via la fonction `ooDocument()` que nous allons voir bientôt) il est nécessaire non seulement d'indiquer un fichier OpenDocument mais aussi de choisir sur quel membre on veut travailler.

3.3 Classes de documents

On sait que, avec le logiciel OpenOffice.org, il est possible d'éditer quatre classes de documents, correspondant respectivement aux quatre modules fonctionnels de la suite bureautique. Le tableau suivant met en correspondance les quatre classes avec les outils et les suffixes de fichiers correspondants :

Fonction	***Module OOo***	Suffixe OOo	Suffixe ODF	***Classe***
traitement de textes	Writer	.SXW	.ODT	text
tableur	Calc	.SXC	.ODS	spreadsheet
présentation	Impress	.SXI	.ODP	presentation
dessin	Draw	.SXD	.ODG	drawing

Ce tableau n'indique que les classes fondamentales de fichiers ; il existe en réalité d'autres possibilités, notamment les modèles de documents et les documents maîtres. En pratique, tout document OOo ou ODF peut être traité via OpenOffice::OODoc comme s'il était de l'une des classes ci-dessus.

Pour OpenOffice::OODoc, les noms de fichiers sont indifférents. Vous pouvez intervertir les suffixes, ou donner des suffixes inconnus pour OpenOffice.org. La classe d'un document, indiquée par l'un des quatre mots-clés de la colonne de droite du tableau, est la seule chose qui compte. Si par exemple vous voulez créer un nouveau document de type dessin, peu importe que vous lui choisissiez ou non un fichier de la forme « xyz.sxd » ; vous devrez indiquer la classe « `drawing` ». En sens inverse, si vous ouvrez par programme un document ODF dont vous ignorez la classe, vous pouvez utiliser une méthode `contentClass()` qui retournera l'un des quatre mots-clés (ne perdez pas de temps à analyser le nom du fichier).

Il est important de comprendre dès à présent qu'il n'existe pas, dans OpenOffice::OODoc, une interface de programmation distincte pour chaque classe de document. Bien entendu, il est impossible d'appliquer la même logique à tous les types de documents, sachant que les éléments ne sont pas agencés de la même manière. Mais les mêmes primitives sont utilisables quelle que soit la classe.

Prenons un exemple simple. Dans un document de classe *text* (Writer), on peut trouver des paragraphes et des tableaux intercalés. Dans un document *spreadsheet* (Calc), tout élément est nécessairement contenu dans un tableau, chaque feuille de calcul étant un tableau. Mais dans les deux cas, tout le contenu est géré via le membre « content », et un tableau est toujours un tableau. Par exemple, pour accéder à la cellule numéro `$C` de la ligne numéro `$L` du tableau `$T` d'un document `$doc`, la méthode est,

```
$cellule = $doc->getCell($T, $L, $C);
```

que `$T` soit une feuille dans un document *spreadsheet* ou un tableau dans un document *text*. La réalité est légèrement plus complexe que ce qui apparaît dans cet exemple (les contraintes et limites de certains accesseurs ne sont pas les mêmes avec toutes les classes de documents) ; mais les méthodes et les types de données sont les mêmes.

3.4 Autres formats de fichiers

OpenOffice::OODoc peut gérer des documents stockés sous d'autres formes que les fichiers ODF, comme par exemple des documents XML en mémoire, des fichiers XML non compressés, ou encore des archives compressées ne contenant que des parties de documents, ou contenant d'autres choses que des documents réguliers (ex: des bases de données de styles ou d'éléments de contenu non éditables, mais utilisables dans des documents réguliers).

En fait, il est possible d'accéder à toutes sortes de documents XML, dont certains peuvent n'avoir aucun rapport avec le format OpenDocument. Rappelons à ce sujet que toutes les méthodes de XML::Twig et d'Archive::Zip sont disponibles à travers OpenOffice::OODoc. Cette possibilité est intéressante, notamment pour les applications d'entreprise, car elle permet, au moyen d'une seule interface, de mettre en relation des messages XML de toutes natures avec des documents éditables.

Mais nous n'en parlons pas plus dans ce guide d'initiation.

4 Usage de la documentation

Le manuel de référence détaillé d'OpenOffice::OODoc étant assez volumineux, il a été découpé par modules. Chacun des ces modules porte un nom de la forme OpenOffice::OODoc::XXX, où XXX peut être considéré comme le « code » d'un thème fonctionnel particulier. Le premier module se nomme simplement OpenOffice::OODoc, mais ce n'est qu'une brève introduction, qui présente seulement quelques fonctions d'initialisation. Les autres sont les suivants :

Module	***Description***
OpenOffice::OODoc::Intro	Petit guide de prise en main. Inutile si vous lisez déjà le présent guide.
OpenOffice::OODoc::Text	Fonctions dédiées au traitement des contenus textuels, quelle que soit la classe de document
OpenOffice::OODoc::Styles	Fonctions concernant la présentation, la mise en page, et tout ce qui passe par des éléments « style »
OpenOffice::OODoc::Image	Fonctions relatives à l'import, l'export et l'intégration d'images dans les documents
OpenOffice::OODoc::Document	Fonctions (peu nombreuses) agissant simultanément dans deux ou plusieurs domaines (texte, image, style)
OpenOffice::OODoc::Meta	Fonctions concernant les métadonnées (spécifiquement dédiées au membre « meta »)
OpenOffice::OODoc::Manifest	Accesseurs permettant de consulter, modifier, créer ou supprimer des entrées dans les descripteurs de fichiers ODF ; leur connaissance n'est pas nécessaire en général
OpenOffice::OODoc::XPath	Fonctions de bas niveau, pour utilisateurs avertis, permettant d'accéder à tous les objets non prédéfinis, notamment parmi les membres *content*, *meta*, *styles* et *manifest*.
OpenOffice::OODoc::File	Fonctions de bas niveau relatives à l'accès physique aux fichiers d'archive compressés ODF

Tous ces modules de documentation sont accessibles en ligne à partir de la page d'accueil du projet au CPAN (http://search.cpan.org/dist/OpenOffice-OODoc).

De plus, ils sont automatiquement installés sur chaque machine avec la distribution, tantôt sous forme de pages HTML (distribution PPM pour Windows), tantôt sous forme de pages de manuel (distribution CPAN pour Unix).

Si la distribution CPAN est installée, et si le « man » d'Unix est disponible, il suffit d'une commande

```
man OpenOffice::OODoc::XXX
```

pour afficher dans la console le chapitre correspondant.

Les pages HTML sont distribuées dans des sous-répertoires hiérarchisés selon le nom logique des modules. Le document OpenOffice::OODoc::XXX est accessible via un chemin de la forme

```
<base>\OpenOffice\OODoc\XXX.html
```

où <base> correspond à la base du chemin d'accès à la documentation Perl, dépendant de l'environnement.

Il n'y a pas de chapitre de manuel correspondant à une classe particulière de document. Le manuel est organisé par catégorie d'opérations.

À l'intérieur de chaque module de documentation, les fonctions et méthodes sont (faute d'une meilleure idée) présentées par ordre alphabétique.

5 Création et enrichissement d'un document

OpenOffice::OODoc est conçu principalement pour consulter et modifier des éléments de contenu dans des documents existants. La création pure de documents à partir de rien n'est pas son application la plus commune et la plus utile. C'est cependant une possibilité, et comme c'est l'une des plus simples à expliquer, commençons par là.

Cette approche est d'autant plus sensée que, une fois l'enveloppe du document créée, les méthodes utilisables pour y mettre du contenu sont les mêmes que celles que nous utiliserons pour ajouter du contenu à des documents préexistants.

5.1 Initialisation

La première chose à faire consiste à instancier un objet *Document* en utilisant la fonction d'initialisation `ooDocument()`, assortis de paramètres nommés qui dépendent de nos intentions. Ensuite, cet objet va être utilisé pour appeler les différentes méthodes de création d'éléments.

Supposons, dans un premier temps, que nous voulions créer un nouveau document (initialement vide) de classe `spreadsheet`, pour déposer ensuite des données dans quelques cellules. Le script complet pourrait être le suivant :

```
use OpenOffice::OODoc;

my $doc = ooDocument
              (
              file      => 'essai.ods',
              create    => 'spreadsheet',
              member    => 'content'
              );

die 'ça ne marche pas !' unless $doc;
my $feuille = $doc->getTable(0, 15, 12);
$doc->renameTable($feuille, 'MaPage');
$doc->cellValue($feuille, 'A1', 'Valeur A1');
$doc->cellValue($feuille, 'L15', 'Valeur L15');
$doc->save;
```

Explication :

La première fonction appelée, le constructeur `ooDocument()`, est renseignée avec trois options (dont l'ordre n'a d'ailleurs pas d'importance). La première est évidente, c'est le nom du fichier cible (attention, tout fichier préexistant portant le même nom sera écrasé !). La seconde option « `create` » indique par sa seule présence que le document n'existe pas encore, donc qu'il va falloir le créer, et cette option appelle tout

naturellement comme valeur la classe du document à créer, soit ici « `spreadsheet` » (le seul fait que le nom du fichier se termine par `.ods` n'est pas pris en compte pour ce choix). Enfin, comme nous avons l'intention d'agir sur le contenu du document (et non, par exemple, sur les styles ou les métadonnées), nous ajoutons une option « `member` » indiquant ce choix (en l'occurrence, j'ai codé explicitement cette option pour des raisons pédagogiques, sachant que, en l'absence d'indication, le constructeur `ooDocument()` sélectionne le membre « `content` » par défaut).

Remarque importante : à l'appel de ooDocument(), la structure du document est construite et prête à l'emploi, mais uniquement en mémoire ; le fichier n'est pas encore créé à ce stade. En cas d'arrêt du programme après cette instruction, il ne subsiste donc aucune trace du document.

En cas d'échec, `ooDocument()` retournerait une valeur nulle, ce qui permet d'agir en conséquence (comme ici).

5.2 Peuplement du document

Nous allons maintenant voir quelques exemples usuels de création ou de modification de contenu, à savoir les tableaux et les paragraphes.

Quand on utilise `ooDocument()` pour créer un document de type « spreadsheet », ce document est initialisé avec une seule feuille de calcul, vide, et nommée par défaut (sans grand effort d'originalité) « `First Sheet` ».

L'instruction suivante

```
my $feuille = $doc->getTable(0, 15, 12);
```

accomplit deux opérations.

D'abord elle nous donne une référence `$feuille` (que j'appellerais un « pointeur » si nous étions en C et pas en Perl) de nature à faciliter l'accès ultérieur aux éléments contenus dans une feuille de calcul.

Ensuite elle initialise un espace de travail de 15 lignes sur 12 colonnes dans cette feuille de calcul. Pourquoi ? Nous y reviendrons dans le chapitre sur les tableaux.

Le premier argument (0) de getTable() indique simplement qu'on veut accéder à la première feuille de calcul. Plusieurs objets sont ainsi accessibles par leur position dans l'ordre du document, la numérotation étant basée sur 0 et non sur 1. On peut aussi donner des numéros négatifs, pour désigner des éléments comptés à partir de la fin (ici, `getTable(-1)` donnerait le même objet que `getTable(0)` puisqu'il n'y a qu'une seule table au départ). Pour `getTable()`, le numéro peut aussi être remplacé par le nom (unique) du tableau, tel qu'il apparaît par exemple dans l'onglet de la feuille de calcul.

L'instruction suivante, on l'a compris, attribue à la feuille un nom différent du nom initial par défaut.

Enfin, deux appels de `cellValue()`, avec à chaque fois en arguments la référence de la feuille, les coordonnées d'une cellule et un texte à inscrire, sont exécutés de manière à mettre en évidence les deux bornes de la zone de travail déclarée.

Les méthodes `getTable()`, `renameTable()`, `cellValue()`, qui concernent toutes des conteneurs de texte, sont décrites dans le chapitre OpenOffice::OODoc::Text du manuel de référence.

La dernière instruction « utile » est `$doc->save`. C'est à ce moment (et pas avant) que le fichier est effectivement créé.

Ici, nous n'avons apparemment agi que sur le membre « content » du document, mais nous avons quand même créé un fichier OpenDocument complet et opérationnel (vous pouvez l'ouvrir tel quel avec votre tableur préféré). En effet, la présence de l'option « `create` » provoque la génération automatique de tous les autres membres obligatoires (`styles`, `meta`, etc.), chargés avec des valeurs par défaut.

Transposons maintenant l'exercice à un document de classe « `text` ». Voici une possibilité :

```
use OpenOffice::OODoc;

my $doc = ooDocument
              (
              file      => 'essai.odt',
              create    => 'text',
              member    => 'content'
              );

die 'ça ne marche pas !' unless $doc;
my $paragraphe = $doc->getParagraph(0);
$doc->removeElement($paragraphe);
my $feuille = $doc->appendTable('MaTable', 15, 12);
$doc->cellValue($feuille, 'A1', 'Valeur A1');
$doc->cellValue($feuille, 'L15', 'Valeur L15');
$doc->save;
```

Dans l'ensemble, la ressemblance est forte. La classe de contenu, indiquée par l'option « `create` », est à présent « `text` ». Mais, pour cette classe de contenu, le constructeur crée un document contenant un premier paragraphe vide, et non pas une première feuille de calcul vide. Je commence par supprimer ce paragraphe ; pour ce faire, il suffit de récupérer la référence du premier paragraphe (numéroté 0) au moyen

d'un `getParagraph()` et d'exécuter un `$doc->removeElement()` sur cette référence (ce qui veut dire « supprimer cet élément du document `$doc` »).

Ensuite, comme aucune table n'existe encore, il faut bien la créer. La méthode `appendTable()` ajoute un tableau (ou une feuille de calcul) au document. S'agissant d'une création, il faut bien entendu préciser les dimensions. OpenOffice::OODoc impose l'attribution d'un nom à tout tableau créé ; en l'occurrence, c'est le premier argument de `appendTable()`. Comme je viens de créer et de nommer explicitement cette table, je n'ai pas besoin ici de la renommer (cela dit, la méthode `renameTable()` fonctionnerait aussi bien dans cette classe de document).

Le reste fonctionne comme dans l'exemple précédent.

Poursuivons sur le thème de la génération document avec un autre exemple, inspiré de l'utilitaire `text2ooo` fourni dans la distribution. Ce programme très court et très simple convertit un fichier de texte plat en fichier OpenOffice.org Writer. On suppose que le nom du fichier source est le premier argument de la ligne de commande, le nom du fichier OOo cible étant le second.

```
use OpenOffice::OODoc;

open SOURCE, "<", $ARGV[0];
my $doc = ooDocument
              (
              file      => $ARGV[1],
              member    => 'content',
              create    => 'text'
              );
while (my $ligne = <SOURCE>)
     {
     chomp $ligne;
     $doc->appendParagraph
              (
              style     => 'Standard',
              text      => $ligne
              );
     }
close SOURCE;
$doc->save;
```

Il ne faut rien de plus pour convertir un fichier plat en fichier ODF ! Cela dit, nous avons ici pris un parti simpliste : chaque ligne du fichier source provoque l'ajout d'un paragraphe, et tous les paragraphes ont le style `Standard`.

5.3 Renseigner les métadonnées

Jusqu'ici nous avons créé des documents avec du contenu, mais sans toucher à leurs propriétés globales (quant aux styles, nous verrons plus loin).

À présent, si nous voulons enrichir ces documents avec des métadonnées appropriées, nous devons les aborder sous un autre angle, en accédant cette fois au membre « `meta` ».

Reprenons notre premier fichier, « `essai.sxc` ». Cette fois, il existe puisque nous l'avons créé. Il ne s'agit donc plus de le créer ; nous allons donc commencer par utiliser `ooMeta()` sans l'option « `create` » (et donc sans indiquer la classe, que OpenOffice::OODoc va découvrir automatiquement) :

```
use OpenOffice::OODoc;

my $doc = ooMeta(file => 'essai.sxc');
```

Notons au passage que `ooMeta()` est un constructeur particulier, spécialement dédié aux métadonnées, et sachant par défaut sélectionner le membre `meta` ; l'argument `member` n'est donc pas nécessaire. En revanche, si nous avions voulu créer le document tout en renseignant ses métadonnées, nous aurions ajouté la même option `create` que précédemment pour `ooDocument()`.

À partir d'ici, nous disposons d'un objet `$doc` qui nous donne accès aux métadonnées mais pas au contenu (ni aux styles, d'ailleurs). Les méthodes propres à cet objet sont présentées dans le chapitre OpenOffice::OODoc::Meta du manuel. La suite est très simple.

Par exemple, pour donner un titre, un sujet, une description et quelques mots-clés, le code ci-dessous ne nécessite pas d'explication laborieuse :

```
$doc->title('Les joies du tableur');
$doc->subject('Initiation à OpenOffice');
$doc->description('Document de travail à compléter');
$doc->keywords('tableur', 'feuille de calcul', 'bureau');
```

Notez au passage que les accesseurs ci-dessus sont bi-directionnels ; ils peuvent être utilisés aussi bien pour consulter la valeur existante que pour la modifier. Par exemple

```
my $titre = $doc->title;
```

retourne le titre du document (s'il existe).

Pour valider et conserver les changements, ne pas oublier de terminer par

```
$doc->save;
```

Mais ici, nous avons utilisé un programme pour créer un fichier OOo et le charger de contenu, puis un autre programme pour lui attribuer des métadonnées. N'est-ce pas un peu laborieux ? Ne pourrait-on pas faire les deux dans la même session, c'est-à-dire ouvrir à la fois « `meta` » et « `content` », et enregistrer le tout en une seule fois par `save` ? Bien sûr que si, mais cela appelle une autre explication.

6 Accès à plusieurs membres d'un même fichier

Dans OpenOffice::OODoc, il existe une séparation technique entre le traitement des documents proprement dit et les opérations sur les fichiers. Quant vous utilisez le constructeur `ooDocument()` avec un paramètre `file`, ce qui se passe est un peu plus complexe que ce que vous voyez. En réalité, deux objets distincts sont créés :

1. l'objet « `Document` » lui-même, le seul qui vous intéresse directement ;
2. un objet « `File` », invisible, qui prend en charge l'accès physique au fichier (qui, je vous le rappelle, est une archive compressée).

Par exemple, la méthode `save` que nous utilisons pour enregistrer le document n'est qu'une méthode « creuse » (un « *stub* », si l'on veut) ; en fait, la méthode `save` de l'objet `Document` ne fait qu'appeler la méthode `save` de l'objet `File` auquel il est associé.

Supposons maintenant que je veuille, dans un même document et dans la même session de travail, exécuter les trois actions suivantes :

1. changer le titre du document, donc accéder aux métadonnées ;
2. ajouter un paragraphe, donc accéder au contenu ;
3. définir un style nommé (visible et réutilisable pour l'utilisateur final) pour ce paragraphe, donc accéder aux styles.

Dans ce cas, je peux évidemment ouvrir et enregistrer trois fois de suite le fichier, pour utiliser successivement les membres `meta`, `content` et `styles`. Mais il est beaucoup plus performant et élégant d'ouvrir le fichier une seule fois et de l'enregistrer une seule fois en fin de traitement. Pour ce faire, nous devons tout simplement créer explicitement un objet `File`, via un constructeur `ooFile()` disponible à cet effet, et le partager entre trois objets `Document`. Le script complet est à la page suivante. En voici l'explication :

D'abord, nous créons un objet `File` en appelant la fonction `ooFile()` qui demande un seul argument, le nom du fichier. Nous gardons cet objet dans la variable `$archive`. Ensuite, nous appelons `ooMeta()` en lui donnant non pas une option file, mais plutôt une option archive. Cette option indique a `ooMeta()` qu'il ne faut pas chercher à accéder directement à un fichier, mais que nous voulons nous « connecter » à une archive déjà ouverte par `ooFile()`. Nous utilisons la même option `archive` pour appeler `ooDocument()` deux fois de suite, une pour le membre `content`, l'autre pour le membre `styles`. Nous avons donc ouvert trois « connecteurs » (dont chacun autorise une catégorie d'opérations) liés au même fichier.

A partir de ce point, chaque objet est utilisé selon sa fonctionnalité. Nous appelons la méthode `title` de l'objet `$meta` pour agir sur le titre (qui est une métadonnée). Nous utilisons ensuite l'objet `$texte`, dont nous savons qu'il a été créé par `ooDocument()` en relation avec le membre `content`, pour créer un conteneur de texte (en l'occurrence un paragraphe) ; ce nouveau paragraphe est affecté d'un style « `EnCouleurs` » qui n'existe pas encore. Nous utilisons enfin l'objet `$deco`, créé lui aussi par `ooDocument()` mais en liaison avec le membre `styles`, pour définir un style de paragraphe d'un beau bleu sur fond jaune (nous parlerons plus loin de la création des styles).

Enfin, un unique appel de `save` sur l'objet partagé `$archive` enregistre tous les changements.

```
my $archive    = ooFile('essai.odt');
my $meta       = ooMeta
                    (
                    archive   => $archive
                    );
my $texte      = ooDocument
                    (
                    archive   => $archive,
                    member    => 'content'
                    );
my $deco       = ooDocument
                    (
                    archive   => $archive,
                    member    => 'styles'
                    );
$meta->title('Coloriages');
$texte->appendParagraph
          (
          style     => 'EnCouleurs',
          text      => 'Paragraphe coloré'
          );
$deco->createStyle
          (
          'EnCouleurs',
          family    => 'paragraph',
          parent    => 'Standard',
          properties =>
               {
               'area'                  => 'text',
               'fo:color'              => odfColor('blue'),
               'fo:background-color'   => odfColor('yellow')
               }
          );
$archive->save;
```

Remarque : l'objet `$archive` (créé ici par `ooFile()`) gère automatiquement l'enregistrement des contenus modifiés par tous les objets créés en relation avec lui par `ooMeta()` ou `ooDocument()` ; il sait ce qu'il doit faire à l'appel de la méthode `save`. D'autre part, ici, on aurait pu remplacer `$archive->save` par, au choix, `$meta->save`, `$texte->save` ou `$deco->save`, car tout appel de `save` sur un objet connecté à un objet `File` équivaut à un appel à l'objet `File` lui-même.

Jusqu'à présent, nous avons utilisé la méthode `save` sans argument. Par défaut, cette méthode réécrit le fichier qui a été indiqué via l'option file à l'appel de `ooDocument` (ou de `ooFile`), à moins qu'elle ne le crée si, en plus, l'option `create` a été utilisée. Il est cependant possible de fournir un nom de fichier explicite, différent de celui du fichier source, de manière à ne pas modifier ce dernier :

```
$doc->save('cible.odt');
```

On peut également appeler plusieurs fois `save` avec des cibles différentes au cours des étapes d'un traitement, de manière à enregistrer des variantes d'un même modèle de document.

Si vous voulez en savoir plus sur les possibilités propres à l'objet `File`, qui permet aussi d'exporter ou d'importer des données brutes sans interprétation ni traitement du contenu, voyez le chapitre OpenOffice::OODoc::File.

7 Traitement de textes

Dans ce chapitre, nous allons explorer quelques possibilités, relatives à la recherche, à la création et à la modification d'éléments textuels. Ceci sans tendre à l'exhaustivité, loin de là. Nous utiliserons comme objet de référence une variable `$doc`, en supposant qu'elle a été créée via `ooDocument()` en relation avec le membre `content` d'un fichier existant ou nouvellement créé.

7.1 Accès aux éléments par position

Nous avons déjà vu comment sélectionner un paragraphe par son numéro :

```
$p = $doc->getParagraph($n);
```

La même opération est possible avec les titres. Ainsi

```
$t = $doc->getHeading($n);
```

Mais comme chacun sait les titres sont hiérarchisés par niveaux (titre 1, titre 2, ...). L'instruction ci-dessus retournera un élément titre selon sa position, tous titres confondus. On peut jouer plus finement et demander le N^{ième} titre parmi les titres d'un certain niveau, en passant une option `level` :

```
$t = $doc->getHeading($position, level => $niveau);
```

D'autres types d'éléments peuvent être sélectionnés d'après leurs positions. C'est par exemple le cas des listes :

```
$lp = $doc->getItemList($n);
```

7.2 Sélection d'éléments par leur contenu

Il est possible de sélectionner un élément, ou une liste d'éléments, selon le texte qu'ils contiennent. Il existe pour cela des méthodes selectXXX agissant indifféremment sur tout type de conteneur, et dont le critère de sélection est traité comme une expression régulière. Par exemple

```
@liste = $doc->selectElementsByContent('Open.*org');
```

renvoie la liste de tous les éléments textuels (paragraphes, listes, titres, tables, etc) conformes au filtre (ici, il s'agit de tout texte contenant une chaîne commençant par « Open » et se terminant par « org »). Cette méthode permet d'ailleurs des opérations beaucoup plus sophistiquées puisque, moyennant des arguments supplémentaires, elle peut aussi effectuer au passage un traitement automatique quelconque sur chacun des éléments conformes. Cette méthode existe aussi « au singulier » :

```
$element = $doc->selectElementByContent($filtre);
```

retourne le premier (ou le seul) élément conforme au filtre.

Notez bien que, si on tient absolument à savoir si un élément sélectionné par son contenu est, disons, un paragraphe, un titre ou une liste à puces, il suffit de le demander :

```
$element = $doc->selectElementByContent('OpenOffice');
print 'Quel beau titre' if $element->isHeading;
```

sachant qu'il existe aussi des méthodes booléennes `isParagraph`, `isList`, etc. On peut aussi utiliser `$element->getLevel` pour connaître le niveau d'un titre, cette méthode retournant `undef` si l'élément n'est pas un titre.

7.3 Sélection d'éléments par noms logiques

Nous avons déjà vu la méthode `getTable()`, utilisée avec un numéro comme critère de sélection. Sachant que chaque tableau dans un document ODF possède un nom unique[11] (choisi par l'utilisateur ou attribué automatiquement), ce nom peut être utilisé à la place du numéro, ce qui permet d'ignorer l'ordre des tableaux dans les documents et de rendre les programmes plus lisibles. Exemples :

```
$t1 = $doc->getTable("Feuille1");
$t2 = $doc->getTable("Tableau1");
```

Ce mode de sélection des tableaux fonctionne quelle que soit la classe du document.

Même chose pour les styles ou les images dont la sélection par nom est même la seule option supportée :

```
$s = $doc->getStyleElement("NomDuStyle");
$i = $doc->getImageElement("Logo");
```

Mais nous parlerons plus loin des styles et des images.

Autres exemples d'objets sélectionnables par leurs noms, les champs variables définis par l'utilisateur, les entrées bibliographiques, les repères (*bookmarks*) :

```
$champ = $doc->getUserField("AdresseClient");
$biblio = $doc->getBibliographyEntry("GEN99");
$signet = $doc->getBookmark("Ici");
```

Pour en savoir plus sur l'utilisation de ces trois derniers objets, voir le manuel de référence (chapitre OpenOffice::OODoc::Text), car nous n'en dirons pas plus dans ce guide introductif.

11 Avec OpenOffice::OODoc, il est bien sûr possible de changer les noms logiques de ces objets... mais il appartient à l'application de veiller à respecter leur unicité. Il est techniquement possible d'enregistrer momentanément plusieurs tableaux, images ou styles portant le même nom, mais à condition de « faire le ménage » avant que le document ne soit édité ou imprimé sous OpenOffice.org par un utilisateur final.

7.4 Actions sur le texte d'un élément

Les méthodes génériques pour lire et modifier le texte d'un conteneur sont `getText()`, `setText()` et `extendText()`.

La première extrait le texte du conteneur. C'est une méthode polymorphe, c'est-à-dire qu'elle découvre dynamiquement le type du conteneur qu'on lui donne à traiter et s'y adapte (dans une certaine mesure). Par exemple, la ligne suivante

```
$texte = $doc->getText($element);
```

retourne tel quel le contenu de `$element` si cet élément est un conteneur simple comme un paragraphe ou un titre. Elle retourne aussi le contenu si `$element` est une cellule de tableau. Si `$element` est une structure composite, comme un tableau ou une liste, `getText()` s'efforce de traduire comme elle peut le contenu correspondant sous la forme d'une chaîne de caractères, selon des règles qui sont d'ailleurs partiellement paramétrables.

La méthode `setText()` est elle aussi polymorphe, mais n'essayez pas de charger un tableau avec un seul appel de `setText()` !

```
$doc->setText($element, $texte);
```

Si `$element` est un conteneur simple (paragraphe, titre, ou cellule), la ligne ci-dessus annule tout contenu précédent et le remplace par la valeur de `$texte`. En revanche, si `$element` est une liste (ordonnée ou « à puces »), la méthode crée un nouvel élément à la fin de la liste et y place le texte donné ; on peut d'ailleurs dans ce cas l'utiliser « en rafale », c'est-à-dire ajouter plusieurs éléments :

```
$doc->setText($liste, 'Eragon', 'Saphira', 'Brom');
```

Sachant que, dans un conteneur simple, `setText()` écrase le précédent contenu, on peut utiliser à sa place `extendText()`, qui concatène un nouveau contenu à la fin du contenu existant.

Il est possible d'agir collectivement sur l'ensemble des contenus textuels sans adresser un élément particulier, notamment via la méthode recherche-remplacement `selectElementsByContent()` dont le premier argument est un filtre de recherche (expression régulière) et le second un texte de remplacement (ou la référence d'un sous-programme retournant un texte de remplacement).

Dans l'exemple ci-dessous, on remplace les chaînes « Ooo », « ooo » et « oorg » par « OpenOffice.org » dans tout le document.

```
$doc->selectElementsByContent
     (
     "(Ooo|ooo|oorg)",
     "OpenOffice.org"
     );
```

7.5 Ajout et suppression d'éléments

Pour ajouter un paragraphe ordinaire à la suite du document, le plus simple est d'utiliser, comme dans l'un de nos premiers exemples, `appendParagraph()`, avec deux arguments nommés indiquant le style et le contenu :

```
$p = $doc->appendParagraph
                (
                style     => 'Ordinaire',
                text      => 'Saphira est une dragonne bleue'
                );
```

La valeur retournée est le nouvel élément paragraphe.

Si l'élément à ajouter est un titre, on utilise plutôt `appendHeading()` avec les mêmes arguments, plus une option indiquant le niveau (sinon, par défaut, le titre créé est de niveau 1) :

```
$t = $doc->appendHeading
                (
                style     => 'Chapitre',
                level     => 2,
                text      => 'Le Désert du Hadarac'
                );
```

Nous avons déjà vu `appendTable()`.

Toutes ces méthodes ne permettent que d'ajouter quelque chose à la fin du document. Pour faire apparaître les mêmes objets n'importe où dans le document, il suffit de remplacer `appendXxx()` par `insertXxx()`, en ajoutant des informations de position.

```
$ici = $doc->selectElementByContent("Eragon avait soif");
$doc->insertParagraph
                (
                $ici,
                position  => 'after',
                text      => "Il chercha un point d'eau"
                style     => 'Standard'
                );
```

La séquence ci-dessus insère un nouveau paragraphe juste après un élément existant (paragraphe ou autre) préalablement sélectionné selon un critère quelconque. Le premier argument de `insertParagraph()` est l'élément avant ou après lequel doit être inséré le nouveau paragraphe. L'option `position` permet de préciser qu'on veut insérer le nouvel élément avant (`before`) ou après (`after`) l'élément de repérage (par défaut, l'insertion est faite *avant*).

La même logique est applicable à d'autres objets. Il existe des méthodes `insertHeading()`, `insertTable()`, etc, acceptant les mêmes arguments que les méthodes `appendXxx()` correspondantes, avec un élément de repérage en premier argument, et une option de position.

Pour aller plus loin, cherchez dans le chapitre OpenOffice::OODoc::Text du manuel tous les noms de méthodes qui commencent par *append* ou par *insert*.

Quant à la suppression d'éléments, vous pouvez vous contenter d'une méthode unique, `removeElement()`, à laquelle il suffit de fournir en unique argument un élément d'un type quelconque précédemment sélectionné.

Notez bien que la suppression d'un élément composite supprime tous les éléments qui dépendent de lui (c'est comme la suppression d'un répertoire dans un système de fichiers arborescent). Par exemple la suppression d'un élément tableau fait disparaître tous les objets (notamment les cellules) qui composent ce tableau. En revanche, la suppression d'un titre (récupéré par exemple via `getHeading()`) n'affecte ni les titres de niveau inférieur, ni les paragraphes, ni les autres objets présentés sous ce titre à l'utilisateur final, car la hiérarchie des titres n'est (dans la logique d'OpenOffice.org) qu'un artifice de présentation et ne se traduit pas par des liens de dépendance dans la structure de données du document. En revanche, attention, lorsqu'on supprime une section, cela fait disparaître tous les objets qu'elle contient, car la section est vraiment un conteneur et non une représentation intellectuelle.

Un élément peut être *ajouté* sans être pour autant *créé* de toutes pièces. Il est possible de copier un élément et d'insérer cette copie ailleurs dans le document. On peut aussi, comme dans l'exemple ci-dessous, insérer dans un document la copie d'un élément existant dans un autre document. Exemple :

```
$element = $doc1->selectElementByContent($filtre);
$doc2 = appendElement($element->copy);
```

L'élément ainsi cloné peut être n'importe quel conteneur de texte, de style ou d'autre chose. On peut donc utiliser des documents existants comme des bases de données de contenus ou de styles utilisables selon une infinité de combinaisons. Je vous laisse

imaginer les possibilités. La fabrication de documents par assemblage est l'une des meilleures utilisations de cette boîte à outils.

7.6 Structuration d'un document

La méthode `makeHeading()` permet de transformer en un titre de niveau choisi n'importe quel élément existant quel que soit son type. Attention, cette méthode doit être appliquée à bon escient ; elle ne peut que provoquer des dégâts si l'élément qu'on lui demande de changer en titre n'a pas la structure appropriée (par exemple s'il s'agit d'un élément composite tel qu'une section ou un tableau). L'utilisation normale de cette méthode consiste simplement à transformer en titres des paragraphes ordinaires. L'exemple suivant montre comment promouvoir au rang de titre de niveau 2 un élément contenant le texte de la rubrique "Structuration d'un document".

```
my $p = $doc->selectElementByContent
          ("Structuration d'un document");
$doc->makeHeading
          (
          element   => $p,
          level     => 2,
          style     => "Niveau 2"
          );
```

Attention, pour produire un résultat visible et satisfaisant, il est nécessaire de fournir non seulement le numéro de niveau (`level`) mais aussi un style de titre adapté (`style`).

Cette méthode est très utile pour structurer a posteriori un document dans lequel l'auteur initial n'a pas jugé utile de créer des titres hiérarchisés, ou encore dont la structure a été perdue pour des raisons techniques (par exemple un fichier exporté en RTF ou en texte plat à partir d'une application bureautique quelconque).

8 Sections et documents composites

Un élément, quel qu'il soit, n'est pas forcément rattaché directement au corps principal du document. Le document peut par exemple contenir des sections, chaque section contenant à son tour des paragraphes, des tableaux ou d'autres objets. Par défaut, les objets créés par appendXXX() sont directement rattachés au corps du document mais il est toujours possible, grâce à une option "`attachment`", de les intégrer à un élément particulier. L'exemple ci-dessous, après avoir recherché une section particulière (identifiée par son nom), ajoute un nouveau paragraphe à cette section.

```
my $section = $doc->getSection("Section1");
$doc->appendParagraph
          (
          text           => "Suite de la section",
          style          => "Standard",
          attachment     => $section
          );
```

La question du rattachement (ou non) à une section ne se pose pas pour les éléments créés via une méthode `insertXXX()`. En effet, tout élément ainsi créé est placé avant ou après l'élément de référence, et implicitement dans le même contexte et au même niveau hiérarchique que lui.

Dans le dernier exemple, un paragraphe est créé dans une section préexistante retrouvée grâce à `getSection()`.

Une liste d'éléments existants peut être déplacée en bloc dans une section, quelle que soit la position d'origine et le type de chacun des éléments, en utilisant `moveElementsToSection()` :

```
$doc->moveElementsToSection($section, @liste);
```

Attention, il ne s'agit pas d'une copie ; chaque élément est coupé et collé dans l'ordre de la liste.

On peut facilement ajouter une nouvelle section à un document à l'aide de la méthode `appendSection()` :

```
$doc->appendSection($nom);
```

Le programmeur choisit librement le nom de la section, à charge pour lui de faire en sorte qu'il n'y ait pas deux sections portant le même nom.

Un section peut être protégée contre l'écriture à l'aide d'une option "`protected`". Attention, il ne s'agit pas d'une protection forte : elle interdit seulement de modifier le contenu de la section sous OpenOffice.org, mais n'interdit rien du tout à un programme

accédant directement (à l'aide d'OpenOffice::OODoc ou autrement) au contenu des fichiers. On peut aussi compléter l'option "`protected`" par une option "`key`" qui, sous OpenOffice.org, imposera la saisie d'un mot de passe à tout utilisateur voulant enlever la protection.

Dans l'exemple ci-après, qui crée une section protégée par mot de passe, le contenu de la variable `$mot_de_passe` n'est pas ce que l'utilisateur devrait saisir pour déverrouiller le contenu de la section. Ce contenu correspond à ce qui sera effectivement stocké dans le fichier, et que OpenOffice.org considère comme la version cryptée du mot de passe. Si le programme place une valeur de clé arbitraire (non calculée par un algorithme semblable à celui d'OpenOffice.org), le mot de passe de déblocage ne sera a priori connu de personne (ce qui est une manière de dire que seul un autre programme pourra déverrouiller la section, et pas un utilisateur final). Mais il est possible d'utiliser comme clé la valeur du mot de passe enregistré dans une autre section (du même document ou d'un autre), ce qui équivaut à propager à une autre section un mot de passe connu, qui a pu être créé via OpenOffice.org :

```
$doc->appendSection
     (
     $nom,
     protected => "true",
     key       => $mot_de_passe
     );
```

Tandis que `appendSection()` crée une section à la fin du document, `insertSection()` la crée quelquepart dans le document, avant ou après un élément quelconque, ou à l'intérieur d'un élément conteneur (une autre section, par exemple). À part cela, les options de création sont les mêmes.

Une section existante protégée (avec ou sans mot de passe) peut être déverrouillée sans autre forme de procès, comme ceci :

```
$doc->unlockSection($nom);
```

De quoi soulager un utilisateur étourdi, qui aurait oublié un mot de passe après avoir trop hâtivement verrouillé un contenu sensible. Mais qui sait ouvrir une porte doit savoir la refermer, aussi existe-t-il une méthode `lockSection()` capable de (re)mettre les protections en place.

Un document peut être constitué non seulement d'un contenu local, mais aussi d'un assemblage de documents accessibles soit par navigation dans un système de fichiers, soit via l'internet. Un document composé en tout ou partie d'une collection de documents indépendants est généralement appelé un *document maître*, et les documents externes incorporés des *sous-documents*. En pratique, l'intégration d'un sous-document dans un

document maître prend la forme d'une section qui, au lieu d'avoir un contenu local, contient simplement un lien qui pointe vers un fichier externe, ce lien étant spécifié par une option "`link`".

```
my $site = "http://jean.marie.gouarne.online.fr";
$doc->appendSection
     (
     "Guide OODoc",
     link => "$site/doc/oodoc_guide.odt"
     );
```

Il suffit d'utiliser `appendSection()` ou `insertSection()` avec un paramètre "`link`" indiquant le chemin approprié. L'exemple ci-dessus (qui fonctionne réellement), peut insérer le présent Guide (avant même de le télécharger) dans un de vos documents au format ODF. Ici, il s'agit d'un lien HTTP mais bien entendu le paramètre "`link`" peut être renseigné avec un chemin d'accès dans le système de fichiers de votre installation ou avec toute espèce de lien correspondant à un protocole géré par OpenOffice.org.

Notez au passage que cette liaison entre documents fonctionne que le document courant soit officiellement qualifié de "*document maître*" ou non.

Une section est un conteneur pouvant héberger une collection très hétérogène d'objets (paragraphes, titres, images, tableaux, listes, autres sections, etc), tandis qu'un élément de type titre (ou *heading*) ne contient que lui-même. Dans une structure OpenDocument, élément titre n'est guère qu'un paragraphe un peu particulier. Par exemple, quand on supprime ou déplace un titre, les paragraphes placés juste après lui restent là où ils sont. Il n'existe pas nativement, dans le format OpenDocument, d'objet contenant, par exemple, tout un chapitre. Les titres n'ont donc rien à voir avec les sections.

Cependant, OpenOffice::OODoc fournit une méthode `getChapterContent()` qui, dans certaines situations, permet de contourner cette éventuelle difficulté. Cette méthode retourne la liste de tous les éléments qui, pour un lecteur moyen, semblent hiérarchiquement rattachés à un titre de chapitre. Les arguments sont les mêmes que ceux de `getHeading()`, mais ce qui est retourné n'est pas un élément *heading*, c'est une liste (dans laquelle l'élément *heading* lui-même n'est d'ailleurs pas inclus).

```
my @elements = $doc->getChapterContent(2, level => 1);
```

L'instruction ci-dessus charge dans `@elements` la liste de tous les éléments qui constituent le contenu du chapitre dépendant du 3ème titre de niveau 1. Cette liste peut ensuite être utilisée pour effectuer un traitement commun sur ces éléments ou pour les transférer ailleurs. On peut par exemple tranférer tout le contenu d'un chapitre dans une section avec `moveElementsToSection()`.

9 Tableaux et feuilles de calcul

9.1 Généralités

Nous l'avons vu au début de ce guide à travers d'autres exemples, on accède à un tableau par `getTable()` et on peut ajouter un tableau à un document par `appendTable()`. Pour insérer un nouveau tableau à une position quelconque choisie, et non à la fin, on utilise `insertTable()`.

Les tableaux se traitent à peu de choses près de la même manière dans les documents Calc que dans les documents Writer. Il existe cependant une contrainte supplémentaire avec les feuilles de calcul de tableur : lors du premier accès par getTable() à l'une d'entre elles, il faut déclarer, en plus du premier argument qui est l'identifiant du tableau, un nombre de lignes et un nombre de colonnes définissant la surface rectangulaire dans laquelle on va travailler. Pourquoi cette contrainte ?

Tout simplement parce qu'une feuille de calcul de tableur possède théoriquement plusieurs centaines de milliers de cellules parmi lesquelles on sait qu'on n'utilisera généralement qu'une petite portion. Pour de basses raisons d'optimisation technique, OpenOffice::OODoc exige que, dans le cas des documents de ce type, on *déclare* les dimensions du rectangle (commençant à la position A1) dans lequel on va travailler[12]. Cette déclaration n'est pas nécessaire dans un fichier texte (Writer), où tous les tableaux ont des dimensions bien définies et généralement modestes, et dont toutes les cellules sont directement adressables. C'est l'une des différences entre les documents « `spreadsheet` » et les documents « `text` » ; cela dit, la même méthode `getTable()`, mais avec deux arguments de moins, est utilisée pour accéder à une feuille de tableur ou à un tableau de texte. Évidemment, cette contrainte propre aux feuilles de tableur serait très pénalisante pour une application ayant simplement besoin d'accéder à la 500ème colonne de la 30.000ème ligne, car il faudrait initialiser un espace de travail de 30000x500 pour n'utiliser qu'une cellule.

Dans un document de classe text (Writer), où chaque tableau a toujours une taille définie, l'application peut l'obtenir par `getTableSize()` :

```
($lignes, $colonnes) = $doc->getTableSize("Tableau1");
```

Cette méthode retourne, dans cet ordre, le nombre de lignes et le nombre de colonnes. Mais attention, dans un document est de type Calc (`spreadsheet`), le résultat de cette

12 Attention, il n'est pas nécessaire de déclarer une surface englobant *toutes* les cellules contenant des données ; la zone déclarée doit simplement englober toutes les cellules (ou les lignes) auxquelles on veut *accéder* dans la suite du programme.

méthode n'est pas significatif, et l'application ne doit se fier qu'aux dimensions de la zone qu'elle a préalablement initialisée avec `getTable()`.

Notons au passage que, dans les tableaux, quel que soit le type de document, on peut adresser une cellule, au choix, par des coordonnées alphanumériques « à la mode tableur » ou par des coordonnées numériques (ligne, colonne) commençant à zéro. Ainsi, 'A1' peut être remplacé par le couple (0,0) et 'L15' par le couple (14,11).

9.2 Agrandissement et rétrécissement d'un tableau

Il est toujours possible d'ajouter de nouvelles lignes dans un tableau. Exemples :

```
$doc->insertRow("Tableau1", 4, position => 'after');
$doc->appendRow("Tableau2");
```

La première instruction insère une rangée immédiatement après la rangée 4 (qui est en réalité la 5ème rangée puisque les positions commencent à 0) du tableau "Tableau1", alors que la seconde ajoute une rangée à la fin du tableau "Tableau2".

Par défaut, la ligne créée par `insertRow()` est une réplique exacte de la ligne donnée comme repère, et celle que crée `appendRow()` est une réplique de la dernière ligne. Il est possible de fournir des paramètres supplémentaires pour changer cela, mais ce sujet dépasse les ambitions de ce petite guide.

La suppression d'une ligne est très simple. Il suffit de sélectionner l'élément correspondant avec `getRow()` et de le supprimer avec `removeElement()`.

```
$ligne = $doc->getRow("Tableau1", 4);
$doc->removeElement($ligne);
```

Cependant, il est plus élégant d'employer la méthode *ad hoc*, `deleteRow()`, qui permet de supprimer une ligne sans avoir à sélectionner explicitement, au préalable, l'élément correspondant :

```
$doc->deleteRow("Tableau1", 4);
```

Il est possible d'insérer une colonne à l'aide de la méthode `insertColumn()`, dans les mêmes conditions (apparentes) qu'une ligne avec `insertRow()`. On peut également supprimer une colonne avec `deleteColumn()`, mais attention, la symétrie n'est pas totale. Dans le format OpenDocument, les cellules sont contenues dans les lignes et non pas dans les colonnes. Les traitements sur les colonnes sont donc en réalité plus élaborés que les traitements sur les lignes, et il peut y avoir des complications dans certains cas, par exemple quand une colonne insérée ou supprimée traverse l'espace occupé par une cellule élargie sur plusieurs colonnes. En tout cas, il ne faut jamais

employer `removeElement()` pour supprimer une colonne de tableau (à moins d'avoir un but très spécial).

Lorsqu'on envisage des opérations répétées sur un même tableau, il est hautement recommandé de commencer par récupérer avant tout la référence du tableau par `getTable()` et d'utiliser cette référence, plutôt que le nom ou le numéro du tableau, dans les méthodes qui accèdent ultérieurement à son contenu. Dans toutes les méthodes dont le premier argument attendu est un tableau, ce tableau peut être identifié au choix par numéro, par nom ou par une référence d'objet tableau.

Sur le chapitre des optimisations, lorsqu'une application a besoin de traiter toutes les rangées d'un tableau et/ou toutes les cellules d'une rangée, il est nettement plus efficace de récupérer d'abord la liste des lignes par `getTableRows()` puis éventuellement , pour chaque ligne, la liste des cellules par `getRowCells()`, pour travailler ensuite sur les éléments référencés à travers ces listes. Pour les grands tableaux, cette approche est plus performante que l'adressage individuel de chaque cellule par le nom du tableau et les coordonnées.

9.3 Accès aux cellules

Pour une compréhension facile de l'adressage des cellules dans les tableaux, nous allons écrire ci-dessous deux programmes, l'un intuitif et l'autre optimisé, effectuant la même chose, à savoir l'écriture d'une valeur quelconque (disons ici un nombre aléatoire entre 0 et 10) dans chaque cellule d'un tableau de 100 lignes sur 16 colonnes.

```
# version non optimisée
for ($i = 0 ; $i < 100 ; $i++)
     {
     for ($j = 0 ; $j < 16 ; $j++)
          {
          $doc->cellValue("Feuille1", $i, $j, rand(10));
          }
     }


# version optimisée
@lignes = $doc->getTableRows("Feuille1");
foreach $ligne (@lignes)
     {
     @cellules = $doc->getRowCells($ligne);
     foreach $cellule (@cellules)
          {
          $doc->cellValue($cellule, rand(10));
          }
     }
```

Ce double exemple montre au passage la souplesse de la méthode `cellValue()`, qui agit sur la valeur d'une cellule désignée dans un cas par le triplet (nom de tableau, ligne, colonne) et dans l'autre par une référence de cellule déjà sélectionnée. Si le premier argument était une référence de ligne de tableau (et non de cellule), alors la méthode attendrait un second argument indiquant la position dans la ligne. Cette méthode admet donc trois modes de désignation d'une cellule (auxquels on peut ajouter deux modes supplémentaires en tenant compte du fait que, si le premier argument indique un tableau, il peut s'agir d'un nom, d'un numéro ou d'un élément tableau déjà connu).

9.4 Opérations sur les cellules

Jusqu'à présent nous avons traité les cellules comme des conteneurs simples, contenant une seule valeur. La réalité peut être plus complexe, pour les cellules de type numérique. Une cellule de texte contient l'équivalent d'un simple paragraphe. Par défaut, `appendTable()` et `insertTable()` créent des tableaux dont toutes les cellules sont des cellules de texte (c'est cependant paramétrable). On peut ensuite agir sur chaque cellule et lui attribuer un type (date, montant en devise, réel flottant, etc) en utilisant une méthode `cellValueType()` non commentée ici. Par ailleurs, dans un tableau, chaque cellule possède un style qui, combiné avec le style de ligne et le style de colonne correspondant à la position de la cellule dans le tableau, détermine de quelle manière elle sera présentée sous OpenOffice.org. La description complète des caractéristiques de présentation d'un tableau est donc très complexe. Tout cela peut être contrôlé avec OpenOffice::OODoc, mais, clairement, ce n'est pas le but. Plutôt que de tout construire par programme (et donc de coder des centaines de paramètres), la bonne démarche consiste à se servir de modèles de tableaux créés en quelques clics avec le logiciel bureautique dans des documents existants. Il est facile d'étendre ou de réduire ces tableaux en insérant ou en supprimant des lignes.

Dans une cellule numérique, OpenOffice.org stocke en réalité deux contenus : la valeur numérique proprement dite d'une part, et la chaîne de caractères éditables exprimant cette valeur. La méthode `cellValue()` autorise, si nécessaire, la fourniture de deux valeurs au lieu d'une, la seconde étant la valeur textuelle. Dans le cas d'un entier, les deux sont généralement identiques, car 1234 s'écrit « 1234 ». Mais ce n'est pas le cas pour d'autres types numériques. Par exemple, les dates étant stockées en format ISO-8601 (AAAA-MM-JJ), pour inscrire la valeur correspondant au 5 mars 2005 dans une cellule de type date, nous pouvons utiliser l'instruction suivante :

```
$doc->cellValue("Feuille1", "B2", "2005-03-05", "05/03/2005");
```

Pour les calculs dans OpenOffice.org, seules comptent les valeurs internes, mais si le programme ne fournit pas en plus une valeur éditable, l'utilisateur qui ouvrira le

document risque d'être déconcerté, le traitement de textes et le tableur n'ajustant pas automatiquement la valeur visible à la valeur interne quand cette dernière a été modifiée à leur insu par un programme indépendant.

Autre exemple : avec les flottants, le séparateur décimal est le point dans la valeur interne, alors qu'il est peut-être la virgule pour l'utilisateur. Etc...

Pour en finir avec ce sujet, ajoutons ici que, indépendamment de sa valeur, de son type et même de son style, une cellule peut être étendue sur deux ou plusieurs colonnes, en recouvrant les cellules suivantes, avec `cellSpan()` :

```
$doc->cellSpan("Feuille1", "B4", 3);
```

L'instruction ci-dessus élargit la cellule B4 aux colonnes C et D, et fait disparaître les cellules C4 et D4.

Pour aller plus loin sur le traitement des cellules, faites quelques essais, et explorez le chapitre OpenOffice::OODoc::Text[13] du manuel de référence, en examinant plus particulièrement les méthodes suivantes :

```
cellValue()
cellValueType()
getCell()
updateCell()
```

Une cellule peut avoir un contenu bien plus complexe qu'une valeur, un type et des attributs de présentation. Elle peut, tout comme une section, contenir d'autres conteneurs, y compris des paragraphes, des titres, des sections et même des tableaux. En utilisant l'option "`attachment`" des méthodes `appendXXX()` précédemment évoquées, on peut très bien créer de tels objets à l'intérieur d'une cellule.

13 J'insiste sur le fait que l'intitulé OpenOffice::OODoc::Text ne veut pas dire qu'il s'agit d'un chapitre consacré exclusivement aux documents de classe `text`. Cela veut dire qu'il concerne les méthodes dédiées aux *conteneurs de texte*. , ce qui inclut, par exemple, les tableaux en général (y compris les feuilles de calcul).

10 Champs variables

Le domaine des champs variables est assez riche est complexe. Dans ce guide, nous n'en verrons qu'une petite partie.

Imaginons un paragraphe contenant le texte suivant :

"Ce document contient nnn pages"

mais avec, bien entendu, un champ dynamique affichant le vrai nombre de pages au moment où le document est édité.

Admettons que ce paragraphe existe déjà, tel qu'il est écrit ci-dessus, quelquepart dans le document, et que nous l'ayons déjà récupéré (par exemple via `getParagraph()`) dans la variable `$paragraphe`. Nous devons remplacer "*nnn*" par le champ approprié. La méthode est la suivante :

```
$doc->setTextField($paragraphe, "nnn", "page-count");
```

Attention, cette instruction va remplacer toutes les occurrences de "*nnn*" par un champ `page-count`. Une seule instruction peut donc créer plusieurs exemplaires de champs.

Remarquez que, en remplaçant l'argument `$paragraphe` par l'élément représentant le corps du document, nous pouvons remplacer "*nnn*" par le compteur de pages dans tout le document (sauf les entêtes et les pieds de pages) :

```
$doc->setTextField($doc->getBody, "nnn", "page-count");
```

Si nous voulons créer un paragraphe qui n'existe pas encore, et qui contienne le même champ variable, nous pouvons évidemment commencer par remplir ce paragraphe avec un texte contenant un "drapeau" ("*nnn*" ou autre chose) et exécuter le remplacement avec `setTextField()` comme ci-dessus. On peut aussi créer le paragraphe et le remplir par morceaux successifs en combinant `appendParagraph()`, `appendElement()`, `extendText()` et une nouvelle méthode `textField()`. Cette dernière crée un champ de texte variable mais sans l'attacher à une position dans le document ; le champ peut ensuite être utilisé dans la construction du contenu d'un élément.

```
my $p = $doc->appendParagraph
                (text => "Ce document contient ");
$doc->appendElement($p, $doc->textField('page-count'));
$doc->extendText($p, " pages.");
```

Ces exemples n'ont présenté que des champs `page-count`, mais il existe de très nombreuses possibilités. Pour connaître la liste exhaustive des types de champs

possibles, mieux vaut consulter la spécification OpenDocument. En voici cependant quelques exemples :

- `page-number` numéro de page courant
- `chapter` chapitre en cours
- `sheet-name` feuille de tableur en cours
- `date` date courante
- `time` heure courante
- `author-name` nom de l'auteur
- `file-name` nom du fichier
- etc.

La plupart des champs de texte peuvent être assortis de paramètres optionnels. Ces derniers sont à fournir à `textField()` ou `setTextField()` après le type de champ. Il est exclu ici de nous étendre sur une longue liste d'options (il est indispensable de se référer à la spécification pour les connaître toutes). L'exemple ci-dessous, qui termine cette rubrique sur les champs de texte, porte sur un champ de type `chapter` ; ce type de champ accepte une option display qui contrôle ce qui doit être affiché, et qui peut être le titre du chapitre (`name`), le numéro (`number`), ou les deux (`number-and-name`) ; on choisit ici la troisième option :

```
$doc->textField('chapter', display => 'number-and-name');
```

11 Images et boîtes de texte

OpenOffice::OODoc permet aux applications d'insérer, de rechercher, de supprimer des conteneurs rectangulaires pouvant héberger, au choix, des éléments graphiques ou textuels, et d'agir sur leur taille, leur position et d'autres propriétés de présentation.

Attention, cette boîte à outils n'est pas une interface de manipulation d'images graphiques (il y a d'autres modules Perl pour ça). Les images ne sont pas traitées ou analysées, elles sont simplement intégrées dans les documents.

Dans ce chapitre, nous parlons des images et des cadres textuels affichés dans le corps des documents, et non pas associés à des styles de pages (en-têtes, pieds ou fonds de page).

11.1 Insérer une image dans un document

Le plus simple est de partir d'un exemple. Nous devons disposer d'une image dans un format supporté par OpenOffice.org (la liste est longue, je ne la reprends pas ici). Prenons par exemple une image JPEG « `eglantine.jpg` », et supposons qu'elle se trouve dans le répertoire « `C:\Images` ».

Intégrer une image dans un document signifie en réalité créer un élément qui fait référence à cette image et à ses caractéristiques de présentation. Par ailleurs, l'image peut ou non être importée dans le fichier (archive) qui contient le document (mais elle peut aussi bien rester là où elle est et être liée par une référence externe, URL par exemple). Pour permettre à OpenOffice.org de placer l'image dans le document, il faut préciser à quoi elle doit être attachée (ou ancrée), et quelle doit être sa taille (la hauteur et la largeur d'affichage sont totalement indépendantes des dimensions originales de l'image).

Dans un premier temps, choisissons d'importer physiquement cette image, de l'ancrer au premier élément de texte contenant « fleurs sauvages » et de l'afficher sur 12 cm de largeur et 9 cm de hauteur.

```
$p = $doc->selectElementByContent('fleurs sauvages');
$doc->createImageElement
          (
          'Églantine des bois',
          description    => 'Espèce commune',
          attachment     => $p,
          size           => '12cm, 9cm',
          import         => 'C:\Images\eglantine.jpg',
          style          => 'MonStyle'
          );
```

Le premier argument est le nom de l'image. Il est obligatoire et doit être identifiant (sous OpenOffice.org, ce nom apparaîtra dans la liste des images visibles à travers le Navigateur). Le paramètre `description` est facultatif. Le paramètre `attachment` indique que l'image sera ancrée à un élément de texte et précise lequel (il s'agit ici du conteneur de texte préalablement sélectionné d'après son contenu). Les dimensions (largeur puis hauteur) sont indiquées sous la forme d'un chaîne de caractères avec une virgule comme séparateur ; l'unité de mesure peut être accolée à chacune des deux coordonnées (« `cm` » et « `mm` » sont supportés et on n'est pas obligé d'utiliser la même unité pour les deux, donc ici on aurait pu écrire `'12cm, 90mm'`). L'option `import` donne le chemin d'accès au fichier et indique en plus que ce fichier graphique doit être physiquement importé, donc stocké dans le fichier OpenDocument ; si on remplaçait `import` par `link`, l'image serait simplement liée par référence, mais non importée.

La dernière option, `style`, est moins évidente. Elle est pourtant parfois requise si on veut que l'image soit présentée de manière appropriée. OpenOffice.org présente toujours les éléments (images ou autres) en utilisant des styles, simples ou complexes. Quand on crée, par exemple, un paragraphe, OpenOffice::OODoc ne nous oblige pas à fournir l'option `style` (cependant recommandée), mais c'est parce que l'outil choisit un style par défaut, sachant qu'un document régulier contient toujours dès le départ au moins un style de paragraphe prédéfini. Ce n'est pas le cas pour les styles d'images. Mais rassurez-vous, il n'est pas nécessaire de savoir construire un style de toutes pièces pour placer une image. Nous disposons en effet d'une méthode `createImageStyle()` qui, même si on ne lui fournit aucun autre paramètre que le nom du style (obligatoire), définit un style par défaut acceptable au moins pour commencer. Donc, pour compléter l'exemple, si « MonStyle » n'est pas défini ailleurs, nous n'avons qu'une instruction à ajouter :

```
$doc->createImageStyle('MonStyle');
```

Cette méthode, utilisée sans option, crée un style de conteneur rectangulaire possédant des propriétés géométriques "raisonnables" et dont on pourra se contenter dans la plupart des cas. Les utilisateurs avertis et exigeants peuvent aller plus loin en choisissant des paramètres supplémentaires ou en construisant des styles complets.

Pour changer un peu, nous pouvons décider d'ancrer l'image à une page, et, si nous ne voulons pas qu'elle soit coincée en haut à gauche de la page, préciser sa position :

```
$doc->createImageElement (
          'Églantine des bois',
          description    => 'Espèce commune',
          page           => 4,
          position       => '6cm, 10cm',
          size           => '12cm, 9cm',
          import         => 'C:\Images\eglantine.jpg',
          style          => 'MonStyle'
          );
```

On voit ici que `attachment` est remplacé par `page`. Pour un document Writer, l'option page indique un *numéro* de page, mais dans un support de présentation ou de dessin (Impress, Draw), ce serait le *nom* de la page, tel qu'il est visible pour l'utilisateur. La position, elle, est indiquée selon la même syntaxe que la taille. Attention, la position est interprétée par le logiciel de traitement de textes en fonction du style[14] (d'où l'importance, parfois, de l'option `style`).

11.2 Agir sur une image existante

Une image se retrouve par son nom, qui est identifiant dans le document.

Pour retrouver, dans une autre application, l'image que nous avons insérée dans l'exemple précédent, il suffit d'écrire :

```
$image = $doc->getImageElement('Églantine des bois');
```

Si on veut traiter systématiquement toutes les images, sans indication de nom, il suffit d'en demander la liste avec `getImageElementList()` :

```
@images = $doc->getImageElementList;
```

Une image (ou une liste d'images), peut être recherchée par d'autres attributs que le nom. Pour en savoir plus, voyez dans le chapitre OpenOffice::OODoc::Image du manuel toutes les méthodes dont le nom commence par `selectImageElement`.

Une fois un élément image obtenu, toutes les méthodes génériques applicables aux éléments sont disponibles (y compris `removeElement()`). Quant aux méthodes spécifiquement dédiées aux éléments image elles acceptent comme premier argument, au choix, le nom visible ou la référence de l'élément. Ces méthodes permettent de consulter ou de modifier la plupart des valeurs que nous avons vues avec `createImageElement()`. Exemples :

14 Par exemple, avec OpenOffice.org Writer, une image ancrée à une page et dépourvue de style est affichée en position centrée en haut de la page, sans tenir compte des coordonnées fournies. Avec un style par défaut créé via `createImageStyle()`, les coordonnées sont effectivement appliquées, avec pour origine le coin supérieur gauche de la page. D'autres définitions de styles permettent d'interpréter les coordonnées d'autres manières (ou de les ignorer).

```
$doc->imageName('Églantine des bois', 'Rose sauvage');
$doc->imageSize('Rose sauvage', '4cm, 3cm');
$doc->imagePosition('Rose sauvage', '5cm, 2cm');
$doc->imageStyle('Rose sauvage', 'Photo1');
#...
```

Vous pouvez aussi créer un lien (ou remplacer un lien existant) de manière à remplacer le contenu d'une image par un autre contenu, local ou distant, qui sera affiché dans le même conteneur :

```
$doc->imageLink('Image1', 'http://fleurs.net/eglantine.jpg');
```

Tous ces accesseurs `imageXXX()`, si vous ne leur passez que le premier argument (identifiant d'image) ne modifient rien et retournent simplement les valeurs courantes.

Quant aux attributs de correction de couleur, de luminosité, de transparence, etc, ils ne sont pas accessibles à travers ces éléments images. Ils sont sous le contrôle des styles.

11.3 Exportation des images

Il est possible d'extraire les images incluses dans les documents pour les utiliser à l'extérieur d'OpenOffice.org, comme fichiers indépendants. Voir à ce sujet les méthodes `exportImage()` et `exportImages()` dans le chapitre OpenOffice::OODoc::Image.

11.4 Boîtes de textes

Ces objets ont beaucoup de points communs avec les images, puisqu'ils dépendent eux aussi de conteneurs rectangulaires possédant des coordonnées, une taille, un attachement, un style. On peut les créer dans des documents de toutes classes via la méthode `createTextBox()` ou `createTextBoxElement()`.

L'exemple ci-dessous crée une zone de texte attachée à l'une des pages d'un support de présentation.

```
my $p = $doc->getDrawPage("Actions en cours");
$doc->createTextBoxElement
          (
          attachment     => $p,
          name           => 'Marketing',
          position       => '3cm, 5cm',
          size           => '12cm, 1.5cm',
          content        => 'Faites-vous connaître'
          );
```

On remarque que le premier argument obligatoire de `createImageElement()`, le nom, est absent. Ici, le nom est fourni via une option `name`, mais il s'agit justement d'une option (recommandée pour permettre de retrouver l'élément plus tard mais facultative). Par ailleurs, il n'y a ni `import` ni `link`, mais une option `content` qui indique tout

simplement le contenu de la boîte. Cette option est assez souple puisqu'elle permet, au choix, de placer de manière directe le texte choisi (comme ci-dessus), ou d'attacher un conteneur de texte déjà existant ou créé explicitement par ailleurs. On peut par exemple réutiliser un paragraphe existant (dans le même document ou dans un autre).

L'exemple suivant équivaut à couper-coller un paragraphe dans une boîte spécialement créée. Le paragraphe 3 est sélectionné puis "coupé" de son contexte (en utilisant la méthode `cut`, non documentée dans OpenOffice::OODoc car elle est héritée de XML::Twig) et enfin "collé" comme contenu de la boîte. La boite elle-même est attachée au paragraphe suivant, sans indication de coordonnées ni de style (avec OpenOffice.org Writer, elle sera normalement placée au début du paragraphe).

```
my $attache = $doc->getParagraph(4);
my $contenu = $doc->getParagraph(3)->cut;
$doc->createTextBoxElement
        (
        attachment     => $attache,
        content        => $contenu,
        size           => '12cm, 1cm'
        );
```

D'autres possibilités, plus ou moins acrobatiques, sont ouvertes. Il est possible de "mettre en boîte" de la même manière des objets plus complexes que des paragraphes (par exemple des listes ou des tableaux). On peut aussi créer une boîte dans un document d'une certaine classe et y insérer un objet prélevé dans un document d'une autre classe (par exemple, prélever un élément dans un document de classe texte et l'insérer sous forme de boîte de texte dans une présentation). Attention toutefois à ne pas exagérer, car un montage trop acrobatique peut créer des problèmes de présentation difficiles à maîtriser.

L'option style est à considérer de la même manière que pour les images. C'est une option, mais on ne peut pas toujours s'en passer (les principes et les problèmes sont à peu près les mêmes). Petite astuce : un style créé pour une image, par exemple via `createImageStyle()`, est souvent applicable tel quel pour une boîte de texte.

Les paramètres d'une boîte de texte ne sont pas plus figés que ceux d'une image. Il est possible de modifier (et aussi, bien sûr, de consulter) certaines caractéristiques d'une boîte existante à l'aide des méthodes `textBoxSize()`, `textBoxCoordinates()`, `textBoxDescription()`, `textBoxStyle()`, `textBoxContent()`. Et une boîte de texte peut être retrouvée par son nom (ou, à défaut, par son numéro d'ordre dans le document) grâce à `getTextBoxElement()`.

12 Supports de présentation

OpenOffice::OODoc n'est pas vraiment un outil de dessin et il serait sans doute très contre-productif d'écrire un programme pour créer de toutes pièces un support de présentation ou un classeur à dessins. Pourtant, l'interface de programmation offre quelques méthodes facilitant certaines manipulations sur ces types de documents.

Les méthodes d'accès aux pages fonctionnent aussi bien sur les documents "Presentation" que sur les documents "Draw". En revanche elles sont inutilisables avec les documents de classe Texte ou Tableur.

On peut sélectionner une page d'après son nom visible (celui qui apparaît dans les onglets en bas de l'écran d'OpenOffice.org), ou encore d'après son numéro d'ordre (compté à partir de zéro) :

```
my $page = $doc->getDrawPage("Objectifs");
```

Une page peut être renommée :

```
$doc->drawPageName("Ancien nom", "Nouveau nom");
```

Bien sûr, une page de présentation peut être copiée comme tout autre élément et sa copie peut être insérée ailleurs dans le document ou dans un autre support de présentation. (Il est même possible de repiquer une page d'un document de présentation dans un document de dessin (draw) et vice versa.)

Imaginez que je dispose d'un support de présentation "menu1.odp" comportant, dans cet ordre, des pages nommées "Entrée", "Plat principal", "Fromage" et "Dessert", et que je veuille composer un autre document, disons "menu2.odp", réutilisant les pages disponibles, mais dans un ordre différent et avec des répétitions.

Le programme ci-dessous présente une manière de le faire. Il ouvre le document existant et crée un autre support de présentation vide. Il supprime la première page du nouveau document (par défaut, un nouveau support de présentation créé par OpenOffice::OODoc contient toujours une page vide). Puis, en utilisant la méthode générique `appendBodyElement()` qui permet d'ajouter toute sorte d'élément à la fin d'un corps de document, il intègre successivement dans le nouveau document des copies de pages sélectionnées par leurs noms dans le document source. Notez bien la méthode `copy` appelée systématiquement à partir des éléments extraits. Ici nous insérons systématiquement des *copies* des pages extraites du document source (le transfert direct des éléments originaux est à éviter dans ce contexte).

Avant d'enregistrer le nouveau document, nous renommons la dernière page insérée ("Dessert"), car nous avons déjà insérée une autre copie de la même page au début du document, et il vaut mieux ne pas avoir deux page possédant le même titre.

```
my $source = ooDocument
          (
          file      => "menu1.odp"
          );
my $cible = ooDocument
          (
          file      => "menu2.odp",
          create    => "presentation"
          );
$cible->removeElement($cible->getDrawPage(0));
$cible->appendBodyElement
          ($source->getDrawPage("Dessert")->copy);
$cible->appendBodyElement
          ($source->getDrawPage("Entrée")->copy);
$cible->appendBodyElement
          ($source->getDrawPage("Fromage")->copy);
$cible->appendBodyElement
          ($source->getDrawPage("Plat principal")->copy);
my $p = $cible->appendBodyElement
          ($source->getDrawPage("Dessert")->copy);
$cible->drawPageName($p, "Deuxième dessert");
$cible->save;
```

Les méthodes relatives aux pages de présentation[15] ne permettent pas de dessiner, mais elles peuvent être efficace pour le développement d'outils de tri de toutes sortes. Elles facilitent notamment l'utilisation de certains documents comme réservoirs de diapositives.

15 Elles sont documentées dans la page de manuel OpenOffice::OODoc::Text.

13 Styles

Un style est un objet décrivant des caractéristiques de présentation applicables à une famille d'objets. On citera seulement ici quatre catégories de styles, parmi d'autres :

- les styles de paragraphes ;
- les styles de textes, applicables à des portions de textes et non à des conteneurs de textes entiers comme les paragraphes ;
- les styles d'images ;
- les styles de pages.

Sans rapport avec cette classification, on distingue les styles « automatiques » des styles « nommés ». Un style automatique est créé spécialement pour décrire la présentation d'un et d'un seul objet ; comme tous les styles, il possède un nom mais ce nom peut changer d'une édition à une autre du document, ce qui rend sa réutilisation difficile et aléatoire[16]. Un style nommé possède un nom stable, que le logiciel ne change que sur commande de l'utilisateur, donc facilement réutilisable. La création et l'exploitation d'un style automatique sont plus simples quand on travaille sur le contenu d'un document avec OpenOffice::OODoc, parce que les styles automatiques peuvent être définis dans le même membre (`content`) que le contenu. Dans l'exemple de code du chapitre précédent, nous avons créé un style « `MonStyle` » relatif à un objet `$doc` à partir duquel nous avons aussi appelé la méthode `createImageElement()` ; cela signifie que `MonStyle` est un style automatique, que l'utilisateur final ne verra pas sous OpenOffice.org, et qui sera certainement renommé si le document est réécrit par le logiciel bureautique[17].

Avec OpenOffice::OODoc, par défaut (mais tout est paramétrable) un style créé à travers un objet `Document` associé à un membre `styles` est en revanche un style nommé. Supposons que je veuille créer un style de paragraphe « MonStyle » et, dans la même session, l'attribuer à un paragraphe (donc à un élément de contenu) je dois procéder comme dans l'exemple du chapitre 6, auquel je ne reviens pas.

Pour le reste, les styles automatiques et les styles nommés se traitent de la même manière.

16 Avec le logiciel bureautique OpenOffice.org, cette réutilisation est impossible. Avec OpenOffice::OODoc, un programmeur averti peut réutiliser un style automatique, voire le transformer en style nommé, mais nous n'en parlons pas ici.

17 Rassurez-vous, ce renommage n'affectera pas la présentation de l'objet lié à ce style ; le lien sera préservé !

13.1 Définition d'un style de paragraphe

Un nouveau style s'introduit par `createStyle()`. Cette méthode exige le choix d'un nom (qui doit être identifiant). Il faut ensuite préciser la famille et, le cas échéant, le style parent si le nouveau style est basé sur un autre style. D'autres options sont disponibles mais non présentées ici. Enfin, une option `properties` (qui est une référence de hachage) indique la liste des caractéristiques de présentation propres au nouveau style (et qui se combinent éventuellement avec celles du style parent).

Par exemple, le style de ce paragraphe en caractères Times 12 italiques gras, bleu sur fond jaune, dans un document OOo 1, est obtenu de la manière suivante :

```
$doc->createStyle
          (
          "NouveauStyle",
          family    => 'paragraph',
          parent    => 'Text body',
          properties     =>
               {
               'style:font-name'        => 'Times',
               'fo:font-size'           => '12pt',
               'fo:font-weight'         => 'bold',
               'fo:font-style'          => 'italic',
               'fo:color'               => odfColor('blue'),
               'fo:background-color'    => odfColor('yellow')
               }
          );
```

Le style `NouveauStyle` est défini comme un style de paragraphe par l'option `family`. De par l'option `parent`, il hérite du style `Text body` (c'est le nom interne du style prédéfini nommé `Corps de texte` dans la version française d'OpenOffice.org). La liste des propriétés qui le distinguent du style parent apparaît sous `properties`. On voit ici, dans l'ordre, le nom de la police de caractères, la taille des caractères, le poids (*bold*), le style (*italic*), la couleur d'avant-plan et la couleur de fond.

Remarque : les noms symboliques de couleurs utilisés ici sont ceux de la table RGB.TXT de la distribution standard *Xorg*. Ces noms peuvent être différents si vous utilisez une autre table RGB, et doivent être remplacés par des valeurs numériques si vous n'avez pas installé de table de couleurs.

Il est important de noter que la définition des styles est un domaine dans lequel OpenOffice::OODoc ne masque pas entièrement les différences sémantiques entre documents OpenOffice.org version 1 et les documents ODF. Dans l'exemple ci-dessus, qui fonctionne avec l'ancien format, certaines propriétés seraient sans effet dans un document ODF. Dans le format OOo 1, toutes les propriétés sont « à plat » dans une

même structure, quel que soit leur domaine, alors que dans l'ODF elles sont classés par domaines. Dans ce dernier format, il est notamment possible que deux propriétés portent le même nom soient présentes dans la même définition de style mais dans des « zones » différentes et avec des effets différents.

Prenons comme exemple le présent paragraphe dont le fond est jaune mais dont le texte est bleu marine sur bleu clair.

En format OOo 1, un style produisant cet effet serait décrit ainsi :

```
$doc->createStyle
     (
     "NouveauStyle",
     family    => 'paragraph',
     parent    => 'Text body',
     properties     =>
       {
       'fo:color' => odfColor('navy blue'),
       'style:text-background-color' => odfColor('sky blue'),
       'fo:background-color' => odfColor('yellow')
       }
     );
```

Avec l'ODF, le même nom désigne l'arrière-plan des caractères et celui du paragraphe :

```
'fo:background-color'
```

Dans ce cas, il est nécessaire d'indiquer, pour chaque propriété, à quelle partie de la description du style elle se rapporte. Dans le cas d'un style de paragraphe, on dira par exemple qu'il y a une partie '`paragraph`' groupant les propriétés du « fond de paragraphe » et une partie '`text`' se rapportant aux caractères qu'il contient. La partie visée est indiquée par une option '`area`' dans la structure '`properties`'. En l'absence de cette option, la partie sélectionnée par défaut dépend de la famille de style ; dans le cas d'un style de paragraphe, c'est la zone '`paragraph`'. L'inconvénient est que, dans notre dernier exemple de code écrit pour un document OOo 1, seule la propriété '`fo:background-color`' sera prise en compte, car les autres portent sur la police de caractères et la présentation du texte, et relèvent donc de la zone '`text`'. Pour produire l'effet recherché, avec un document ODF, il faudrait réécrire l'exemple comme ci-après.

```
$doc->createStyle
     (
     "NouveauStyle",
     family    => 'paragraph',
     parent    => 'Text body',
     properties     =>
          {
          'fo:background-color' => odfColor('yellow')
          }
     );
$doc->updateStyle
     (
     "NouveauStyle",
     properties     =>
          {
          area           => 'text',
          'fo:color'     => odfColor('navy blue'),
          'fo:background-color' => odfColor('sky blue')
          }
     );
```

On notera que, comme `createStyle` ne permet de paramétrer qu'un seul domaine de propriétés (ici, par défaut, '`paragraph`'), il faut faire appel à `updateStyle` (cette fois avec une option '`area`' dans la structure '`properties`' parce qu'on vise la zone '`text`' qui n'est pas la zone par défaut)[18].

Notons que ce dernier exemple fonctionne avec les documents ODF *et* avec les documents OOo 1, car dans ce dernier cas l'option '`area`' est ignorée.

OpenOffice::OODoc ne fait que donner accès aux propriétés, sans interprétation. Un style peut avoir de très nombreuses propriétés (selon sa famille). Pour connaître la liste complète des propriétés et des valeurs associées, il faut consulter les spécifications du format ODF ou OpenOffice.org[19].

Pour découvrir (et au besoin modifier) les propriétés utilisées dans des styles existants, on peut utiliser la méthode `styleProperties()`. Cette méthode retourne un hachage (comparable à celui que nous avons utilisé sous l'option properties dans l'exemple précédent) indiquant toutes les propriétés décrivant le style et leurs valeurs respectives. L'exemple suivant retourne les propriétés de « MonStyle » tout en ajoutant (ou en modifiant) au passage les propriétés « taille de police » (`fo:font-size`) et « style de police » (`fo:font-style`).

18 Il existe plusieurs autres manières, dont certaines plus performantes, pour produire le même résultat, mais l'optimisation avancée n'est pas l'objectif de ce guide.

19 http://xml.openoffice.org

```
%prop = $doc->styleProperties
          (
          'MonStyle',
          'area'              => 'text',
          'fo:font-size'      => '14pt',
          'fo:font-style'     => 'italic'
          );
```

Nous avons fourni ici une option '`area`' qui précise que les propriétés recherchées appartiennent à la zone '`text`' si notre document est un ODF. Ce sélecteur sera ignoré dans le cas d'un document OOo 1.

Disons quand même quelques mots sur le contenu des propriétés de couleur. Une valeur de couleur est enregistrée par OpenOffice.org en format hexadécimal RVB sur 7 caractères. Le premier caractère est toujours le symbole dièse (#) ; ensuite viennent trois valeurs codées chacune sur deux chiffres hexadécimaux, représentant le rouge, le vert et le bleu. Dans l'exemple précédent, on pourrait coder directement `#0000ff` pour le bleu et `#ffff00` pour le jaune. Mais pour éviter des maux de têtes aux programmeurs, OpenOffice::OODoc fournit une fonction `odfColor()` qui convertit en format hexadécimal compatible OpenDocument des noms symboliques de couleurs, selon une table de correspondance normalement chargée à partir d'un fichier au format RGB standard. D'où l'option relative au fichier de couleurs vue au chapitre sur l'installation. Toutefois, quelques couleurs de base (dont `blue` et `yellow`) sont prédéfinies même en l'absence de fichier de couleurs. Pour mémoire, il existe aussi une fonction `rgbColor()` qui, en se basant sur la même table de correspondance, restitue en clair les noms correspondant à des couleurs connues. (Cela dit, comme le jeu de couleurs est sur 24 bits, je doute que quelqu'un puisse disposer d'un fichier de couleurs exhaustif.)

13.2 Définition d'un style pour une zone de texte

On peut vouloir attribuer à une portion de texte certains attributs de présentation qui la distinguent du reste du paragraphe. Par exemple ici je distingue ce *mot* en l'écrivant en italiques sur fond jaune. Techniquement, cela signifie que la chaîne de caractères concernée se voit attribuer un style automatique créé spécialement pour elle. Bien que ce soit transparent pour l'utilisateur, cela implique pour le programmeur la création d'un style automatique approprié. Ce style sera un style de texte et non un style de paragraphe, car nous ne voulons pas l'appliquer au paragraphe. L'exemple qui suit montre comment le créer.

```
$doc->createStyle
          (
          'ItaliqueJaune',
          family          => 'text',
          properties      =>
               {
               'fo:font-style'               => 'italic',
               'style:text-background-color' => '#ffff00'
               }
          );
```

Le style défini ici est de la famille texte. Nous pourrons donc l'appliquer à autre chose qu'un conteneur complet tel qu'un paragraphe.

Remarquez au passage que la couleur d'arrière-plan est désignée par une propriété `style:text-background-color` et non `fo:background-color`, car l'arrière-plan d'un caractère n'est pas la même chose que l'arrière-plan d'un paragraphe. D'autre part, on a choisi ici (pour changer par rapport à l'exemple précédent) de noter la valeur de couleur directement en format OOo sans passer par la fonction de traduction de couleur symbolique.

Mais comment faire pour appliquer ce style ? C'est simple, il faut d'abord de sélectionner (par exemple via l'une des méthodes de recherche que nous avons vues) le conteneur dans lequel nous voulons appliquer ce petit coloriage, puis invoquer la méthode `setSpan()` pour mettre en association une zone ou une étendue (*span*) de son texte avec le style que nous venons de créer.

Pour appliquer ce style de texte à toutes les chaînes « *Office* » présentes dans un paragraphe `$p`, l'appel est le suivant :

```
$doc->setSpan($p, 'Office', 'ItaliqueJaune');
```

Le second argument de cette méthode peut être un filtre plus ou moins complexe, et pas seulement une chaîne définie de manière exacte.

Il est possible, grâce à la méthode `removeSpan()`, de supprimer tous les effets de style intérieurs à un conteneur de texte. Par exemple

```
$doc->removeSpan($p);
```

supprimerait non seulement les effets du `setSpan()` précédent, mais également tous les autres « *spans* » éventuellement placés par ailleurs dans l'élément `$p`.

13.3 Définition d'un style d'image

Prenons maintenant un second exemple, la création d'un style d'image :

```
$doc->createStyle
          (
          "MesPhotos",
          family         => "graphic",
          parent         => "Graphics",
          properties     =>
               {
               'style:vertical-pos'    => 'from-top',
               'style:horizontal-pos'  => 'from-left',
               'style:vertical-rel'    => 'page',
               'style:horizontal-rel'  => 'page',
               'draw:luminance'        => '4%',
               'draw:contrast'         => '2%',
               'draw:gamma'            => '1.1',
               'draw:transparency'     => '5%',
               'draw:red'              => '-3%',
               'draw:green'            => '2%',
               'draw:blue'             => '1%'
               }
          )
```

Ce style est de la famille `graphic` et il hérite du style de base de cette famille, `Graphics` (avec un grand `G` et un `s` à la fin). C'est le cas habituel pour un style d'image. Nous lui attribuons ensuite un riche assortiment de propriétés. Les premières sont ici des propriétés «`style`» relatives au mode de placement de l'image ; elles indiquent selon quelles règles doivent être interprétées les propriétés de position qui appartiennent aux objets image et que nous avons déjà vues. Ici, on précise que les coordonnées sont relatives à la page et que leur origine est le coin supérieur gauche. Quand aux propriétés « `draw` », elles indiquent des corrections de luminosité, de contraste, d'indice gamma, de transparence et de couleurs (les accros de la photographie et de la retouche d'images s'y retrouveront).

13.4 Mise en page

La mise en page se fait au travers de styles de pages. Il vaut mieux savoir que ces styles (qui traitent notamment les en-têtes, pieds de pages et fonds de pages) sont également pris en charge par OpenOffice::OODoc, mais que leur paramétrage est assez complexe. Plus généralement, même pour des utilisateurs maîtrisant bien la logique de cette interface de programmation, il est préférable, autant que possible, de s'appuyer sur des modèles de documents contenant des styles de pages appropriés plutôt que de tout créer par programme.

Avant tout, il faut éviter de chercher un moyen direct tel que « appliquer tel style à la page numéro tant ». Ce moyen n'existe pas. Compte tenu des hasards qu'impliquent les possibilités d'insertion et de suppression d'éléments, les variations possibles du nombre de pages d'un document, voire les options de numérotation des pages (cette numérotation ne commence pas forcément à 1 et ne commence pas forcément à la première page physique du document), la logique d'OpenDocument exclut l'existence d'un lien persistant entre numéro de page et style de page. En fait, les styles de pages sont attachés à des éléments de contenu, par exemple des paragraphes, selon une technique assez simple : certains styles de paragraphes possèdent des marques spéciales signifiant «à partir de ce paragraphe et jusqu'à nouvel ordre, les pages ont tel style ».

En pratique, il suffit d'appliquer au premier conteneur de texte un attribut `style:master-page-name` qui déterminera le style de la première page et de toutes les pages suivantes au moins jusqu'au premier saut de page. Par défaut, OpenOffice.org Writer attribue le style de page « Standard ». Attention, le style de paragraphe par défaut s'appelle aussi « Standard », mais il ne s'agit pas du même type de styles. Un style de page est, dans le vocabulaire OpenDocument, un « `master-page` » ce qui, en Français, pourrait se traduire par « patron de page » (patron étant à prendre ici au sens de *Modes et Travaux*).

Un « `master-page` » détermine la structure d'une page, et notamment la présence éventuelle d'une entête et/ou d'un pied de page. Mais il ne contient pas d'indication sur la géométrie de la page (hauteur, largeur, marges). Chaque « `master-page` » contient une référence à un autre objet, nommé « `page-layout` » (présentation de page)[20]. En conséquence, si on veut créer de toutes pièces un style de page, il faut créer d'une part une présentation (*layout*) et d'autre page une structure utilisant cette présentation (*master*). Ces deux opérations peuvent être réalisées avec les méthodes `createPageLayout()` et `createMasterPage()`. Il est évidemment permis de ne créer qu'un « `master-page` » qui réutilise un « `page-layout` » existant (une présentation de page est justement faite pour être partagée, ce qui permet par exemple de créer des pages ayant la même géométrie mais des contenus différents dans les entêtes et les pieds de pages). On peut aussi modifier des *masters* et les *layouts* existants grâce à `updateMasterPage()` et `updatePageLayout()`.

Enfin, une méthode `masterPageExtension()` permet d'attacher une entête ou un pied de page à un *master* déjà défini.

20 Dans l'ancien format OpenOffice.org 1.0, le « `page-layout` » se nommait « `page-master` ». On avait donc affaire à des « `master-pages` » se référant à des « `page-masters` », ce qui ne manquait pas de créer des confusions.

Il est recommandé de créer les styles de page dans l'espace « `styles` » et non dans l'espace « `content` ».

Mais ce n'est pas tout. Définir un style de page et sa présentation est une chose, l'appliquer dans le corps du document en est une autre. Le saut de page est la modalité la plus simple. Il existe une méthode `setPageBreak()`, qui s'applique à un paragraphe, et dont l'une des options permet de dire « à partir de ce paragraphe, on utilise tel style de page ». Cette méthode est documentée dans OpenOffice::OODoc::Document, et je n'en dis pas plus à son sujet ici. Elle est utilisable notamment de la manière suivante :

```
$doc->setPageBreak
        (
        $para,
        style     => "xyz001",
        page      => "NouveauStyleDePage"
        );
```

Pour des explications sur l'option « `style` », voir la documentation ; disons simplement ici que cette option est obligatoire et doit contenir un nom unique choisi arbitrairement. L'option « `page` », plus significative, indique quant à elle le nom (unique) d'un « `master-page` » existant ou qui sera créé par le programme. Le premier argument, `$para`, est un paragraphe (ou un titre) préalablement sélectionné ou inséré par le programme. Cette instruction réalise deux choses : elle place un saut de page avant le paragraphe, et elle applique au document, à partir de ce paragraphe, un style de page donné. Ce style de page sera appliqué par défaut à toutes les pages suivantes, jusqu'à la fin du document ou jusqu'à la rencontre d'un autre paragraphe traité par un autre `setPageBreak()` comportant une option « `page` ».

Mais tout cela est un peu compliqué au premier abord. Les méthodes citées ci-dessus (par ailleurs plus précisément documentées dans le chapitre OpenOffice::OODoc::Styles), seront peut-être plus compréhensibles à travers un exemple. Le programme ci-dessous utilise les espaces « `content` » et « `styles` » d'un fichier OpenDocument. A l'aide de la méthode `pageLayout()`, il récupère le nom de la présentation (*layout*) utilisée par le style de page (*master*) « Standard » (ceci pour ne pas alourdir l'exemple en créant un `page-layout` complet). Il crée un style de page (`master-page`) « NouveauStyleDePage ». Utilisant `masterPageExtension()` avec le sélecteur « `footer` », il attache au nouveu *master* un pied de page contenant un paragraphe préalablement créé et contenant le mot « Page » suivi d'un champ de numéro de page (`page-number`). Tout ceci a été réalisé dans l'espace de travail « `styles` ». Revenant à l'espace « `content` », on ajoute un nouveau paragraphe de

texte, auquel on applique aussitôt un `setPageBreak()` mettant en oeuvre le nouveau style de page.

```
          # ouverture de l'archive
my $fichier = ooFile("essai.odt");
          # accès au contenu
my $contenu = ooDocument
              (
              archive   => $fichier,
              member    => "content"
              );
          # accès aux styles
my $deco = ooDocument
              (
              archive   => $fichier,
              member    => "styles"
              );
          # construction d'un style de page
          # réutilisant la présentation standard
my $layout = $deco->pageLayout("Standard");
my $master = $deco->createMasterPage
     ("NouveauStyleDePage", layout => $layout);
          # attachement d'un pied de page
my $pied = $deco->createParagraph("Page ");
$deco->appendElement($pied, $deco->textField("page-count"));
$deco->masterPageExtension($master, "footer", $pied);
          # saut de page et mise en service
          # du nouveau style de page
          # dans le contenu du document
my $para = $contenu->appendParagraph
              (
              text      => "Nouvelle page ici",
              style     => "Standard"
              );
$contenu->setPageBreak
              (
              $para,
              style     => "xyz",
              page      => "NouveauStyleDePage"
              );
          # enregistrement des changements
$fichier->save;
```

On pourrait aussi, dans le cadre d'un même style de page, établir une différence entre pages de droite et pages de gauche, ajouter une image d'arrière-plan, modifier complètement la présentation via `updatePageLayout()`, et bien d'autres choses encore, mais tout ne peut pas être présenté ici.

14 Conclusion

J'espère que ce guide facilitera à certains la découverte pratique de la manipulation des documents par programme (et, accessoirement, contribuera à illustrer les avantages que procurent les formats documentaires ouverts, qui permettent d'exploiter des documents bureautiques indépendamment du logiciel qui les a créés).

Pour ceux/celles qui veulent aller plus loin, il faut bien sûr affronter le manuel de référence de la distribution. En complément, l'examen du XML, voire la consultation de la spécification OpenDocument peuvent être utiles pour étendre les possibilités de l'interface. De plus, il ne faut pas oublier XML::Twig, dont les méthodes (puissantes, simples d'utilisation et bien documentées) sont directement applicables aux éléments traités par OpenOffice::OODoc.

Les remarques constructives, les corrections, mais aussi les retours d'expérience sont les bienvenus.

Le forum OpenOffice::OODoc du CPAN est ouvert à toute discussion, donc n'hésitez pas à vous y inscrire si vous avez quelque chose à dire... en Anglais. L'adresse est

http://www.cpanforum.com/dist/OpenOffice-OODoc

Ne vous privez pas non plus de signaler les bogues pour permettre à cet outil de s'améliorer. Il est toujours possible de le faire de manière informelle mais, si la communication en Anglais ne vous décourage pas, passez de préférence par le point d'entrée réservé à cet effet :

http://rt.cpan.org/NoAuth/Bugs.html?Dist=OpenOffice-OODoc

Enfin, les contributions susceptibles d'enrichir la distribution, la documentation associée et même ce guide sont les bienvenues.

Bon courage.