OpenOffice.org

Dans la jungle de l'API OpenOffice.org

Révision 0.3.1 – 7 juillet 2004

Réalisé avec : OOo 1.1.0 **Plate-forme / Os : Toutes**

Distribué par le projet Fr.OpenOffice.org

Table des Matières

1 Introduction

1.1 Premières macros

Lorsque l'on débute dans l'écriture des macros, le plus simple et le plus profitable est d'étudier et de modifier des macros déjà existantes. Il existe de nombreux exemples de macros disponibles parmi lesquels :

- le "Manuel de programmation basic SO 7" disponible ici http://api.openoffice.org/TipsAndTricks/external.html ou là http://docs.sun.com/db/doc/817-3917?l=fr&q=StarOffice&s=t ,
- le " Staroffice Programmer's Tutorial " que l'on peut télécharger ici http://api.openoffice.org/basic/man/tutorial/tutorial.pdf ,
- et le meilleur document que je connaisse, le how-to de Bernard Marcelly, " L'API Openoffice (presque) sans peine " que l'on trouve sur le site français d'Openoffice.org (http://fr.openoffice.org/Documentation/Index.html) Ce how-to a le mérite d'être en français, progressif et bien commenté, et surtout il vous dispense de vous plonger dans le fameux API !

1.2 Consulter l'API

C'est sur cette invite que s'achève le how-to de Bernard Marcelly.

Avant de poursuivre, relisez la partie correspondante, elle contient des éléments intéressants, en particulier la hiérarchie des répertoires et sous-répertoires de la documentation de l'API.

Il faut bien sûr que vous puissiez accéder à l'API lui-même qui fait partie du SDK l'on trouve ici (http://www.openoffice.org/dev_docs/source/sdk/). C'est une immense arborescence qui contient des fichiers html.

1.3 Genèse de ce how-to

J'ai tenté de consulter l'API à de nombreuses reprises et me suis enfuie à chaque fois : je n'y comprenais rien.

Récemment, Tony Galmiche a demandé de l'aide pour une macro, sur la liste prog@fr.openoffice.org (pour s'inscrire :envoyer un mail à prog-subscribe@fr.openoffice.org),et je me suis replongée à nouveau dans l'API. Non que je sois une programmatrice chevronnée, je débute, mais parce que c'est justement en cherchant qu'on apprend. Et là j'ai tenu bon et ai enfin commencé à y voir clair. Pour l'anecdote, cinq heures plus tard, j'avais la réponse au problème posé et elle tenait en deux lignes.

C'est ce périple dans l'API qui m'a conduite à écrire cet how-to. C'est du vécu !

1.4 Comment est organisé ce how-to

Il correspond à mon cheminement personnel à travers l'API. Je suis bien consciente que tout le monde n'a pas les mêmes modes de pensée. Mais il doit bien y avoir quelques francophones, épris d'Openoffice.org et souhaitant écrire des macros qui fonctionnent comme moi !

Le document comprend deux parties :

a) Première partie

On dispose d'une macro et on cherche les correspondances dans l'API.

En quelque sorte, on a un itinéraire (la macro) et des cartes (l'API). On cherche à retrouver l'itinéraire sur les cartes. Ce qui permet d'apprendre à lire une carte.

On prendra une macro qui utilise le curseur d'écriture.

b) Deuxième partie

On a un but de voyage (une idée de macro) et on cherche l'itinéraire (la macro) au moyen des cartes dont on dispose (l'API). Ce qui revient à apprendre à utiliser une carte.

On partira du problème posé par Tony Galmiche : écrire un texte à la fin d'une section.

1.5 Quelques précisions complémentaires

a) Langage de programmation

L'API n'est pas un langage de programmation. C'est un ensemble de propriétés et méthodes utilisables par le programmeur, qu'il programme en Java, C++, Basic ou autre.

Dans ce how-to je me limite au Basic.

Un point important et qui m'a longtemps bloquée : le Basic permet l'utilisation de raccourcis d'écriture et de pseudo-propriétés qui sont présentées en fin de document dans " L'API Openoffice (presque) sans peine ". L'écriture de la macro est certes simplifiée mais la compréhension de l'API est rendue plus difficile.

On trouve des portions de macro comme celle-ci :

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

J'ai longtemps cherché, dans l'API, une propriété nommée String concernant un texte, je ne l'ai jamais trouvée. En effet, en utilisant strictement l'API, on aurait dû écrire ceci :

```
txt = UneCellule.getString()
UneCellule.setString("Bonsoir")
```

Et là on a une chance de trouver la méthode getString() ou setString().

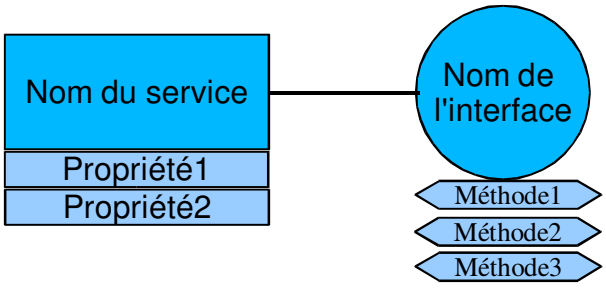
Il faut donc toujours avoir cela présent à l'esprit lorsqu'on décortique une macro, comme nous allons bientôt le faire.

b) Services, Interfaces, Propriétés, Méthodes : le minimum à savoir

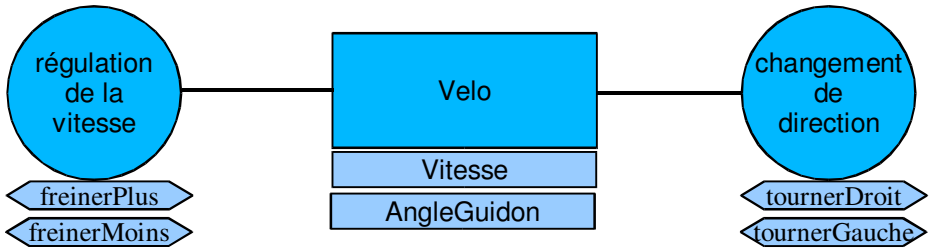
Si vous ne connaissez pas la programmation objet, voici ma propre version minimaliste de la " chose ". Comme je suis autodidacte et débutante, comme vous peut-être, il faut considérer ce qui suit comme de l'à peu près. Les pros vont sans doute tiquer.

Dans l'API on trouve des services et des interfaces attachées aux services.

Un service peut avoir des propriétés, une interface fournit des méthodes permettant de modifier les propriétés.



Par exemple Velo (service) dispose d'un dispositif de changement de direction (interface) qui me permet de tournerDroit (méthode) ou tournerGauche (méthode) et d'un dispositif de régulation de la vitesse (interface) avec lesquels je peux freinerPlus (méthode) ou freinerMoins (méthode). Avec les méthodes tournerGauche, tournerDroit, freinerPlus et freinerMoins je peux modifier les propriétés Vitesse et AngleGuidon de Velo.



Dans les documents sur l'API, on parle aussi beaucoup d'objets. Je crois avoir compris que MonVelo à moi, celui que je prends pour aller faire les courses, est un objet. Et que objet et service c'est quasiment la même chose. Bref, il me semble que je suis autorisée à dire que MonVelo à moi (objet) est un Velo (service). Osons appeler un chat un Chat !

Les méthodes s'utilisent comme ça :

```
MonVelo.tourneGauche()
```

Entre les parenthèses, il faut parfois mettre des informations – on appelle ça des paramètres.

Par exemple :

```
MonVelo.tourneGauche(45)
```

pour tourner de 45 degrés vers la gauche.

C'est l'API qui nous dit quels paramètres sont autorisés ou obligatoires.

2 Décortiquer une macro en fouillant dans l'API

2.1 La macro à analyser

Cette macro, pas très utile j'en conviens, permet de remplacer les trois dernières lettres d'un texte par le mot " Fin ". La voici.

```
1  Dim MonDocument As Object, MonTexte As Object, MonCurseur As Object
2  MonDocument = ThisComponent                          'récupère le document
3  MonTexte = MonDocument.Text                          'récupère le texte du document
4  MonCurseur = MonTexte.createTextCursor               'crée un curseur
5  MonCurseur.gotoEnd(false)                            'curseur à la fin du texte
6  MonCurseur.goLeft(3,true)                            'sélection des 3 derniers caractères
7  MonCurseur.String = "Fin                             'remplace les 3 derniers caractères par Fin
```

Elle suppose, pour fonctionner, qu'un fichier writer soit ouvert et qu'il contienne du texte.

2.2 Par où commencer ?

Comme les deux premières lignes se retrouvent dans toutes les macros qui concernent un document contenant du texte, je ne rechercherai pas la partie de l'API correspondante.

Puisque MonTexte est un objet texte, la recherche commencera avec le service \com\sun\star\text\Text et il s'agit, en progressant à partir de là, de comprendre les lignes 4 à 7.

2.3 Analyse de la macro

a) Service Text

On y a accède, dans l'API, par le chemin suivant : \com\sun\star\text\Text.html.

En cliquant sur Use dans le bandeau en haut de la page \com\sun\star\text\Text.html, on obtient la page \com\sun\star\text\Text_xref.html qui nous intéresse aussi.

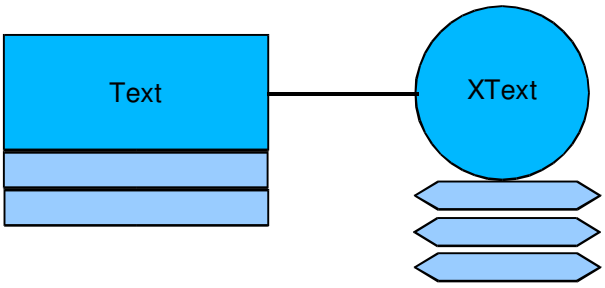
Tous les chemins d'arborescence sont bâtis sur ce modèle. Je ne les préciserai donc plus par la suite.

Dans ces deux pages il y a beaucoup d'informations. Je tente de ne me préoccuper que de celles qui semblent avoir un lien avec la macro.

Ci-dessous les informations que l'on trouve dans l'API et mes commentaires de néophyte :

Service Text		***Commentaire***
Exported Interfaces	**....** **XText** **...**	On pourra donc utiliser les méthodes de XText pour MonTexte.
Services which Include this Service	**::com::sun::star::sheet::SheetCell**	Manifestement ça ne me concerne pas. Mais il n'est pas étonnant de trouver quelque chose comme ça, puisqu'une cellule de feuille de classeur peut contenir du texte. Et puis tiens, eux ils mettent deux doubles points dans le chemin des modules. Bizarre.

Donc l'affaire se présente ainsi :



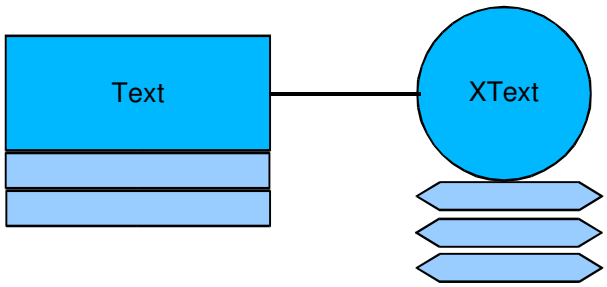
Je vais explorer cette interface XText. Peut-être trouverai-je une méthode createTextCursor.

b) Interface XText

Interface XText		***Commentaire***
Base Hierarchy	**::com::sun::star::uno::XInterface** **XTextRange** **XSimpleText** **XText**	On dirait que XText est une sous-interface de XSimpleText, elle-même sous-interface de XTextRange. Mais quelles conséquences ?
Description	**extends a XSimpleText by the capability of inserting XTextContents.**	extends ??? XSimpleText ???
Methods' Summary	**insertTextContent** **removeTextContent**	Hum ... pas de ça dans ma macro.
Services which Support this Interface	**...** **::com::sun::star::drawing::TextText** **...**	Service which Support ... ??? Là, je comprends que quand on ne précise pas le chemin (cas de Text), c'est qu'on est dans le même module que XText (c'est à dire dans le module com/sun/star/text/, écrit à ma façon). Autrement il faut préciser (cas de ::com::sun::star::drawing::Text)

C'est souvent ça l'API. On croit y trouver des informations, on en ressort avec des questions. J'espère que ça s'éclaircira par la suite.

Pour l'instant je n'ai acquis qu'un peu de vocabulaire (les portions en *italiques* sont en anglais, telles qu'on les trouve dans l'API) :



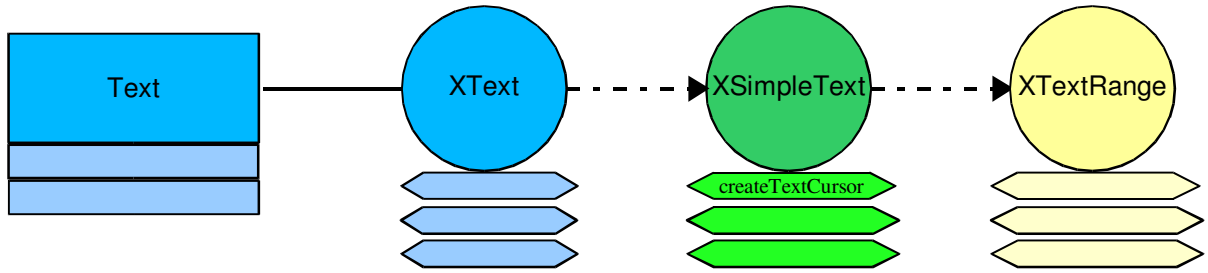
Se lit :
Le service XText supporte (*supports*) l'interface Text.
L'interface XText est une interface exportée (*exported interface*) du service Text.

A part ça, je suis perplexe : pas de méthode createTextCursor !
Il ne me reste plus qu'à aller voir du côté de XSimpleText.

c) Interface XSimpleText

Interface XSimpleText		***Commentaire***
Base Hierarchy	**::com::sun::star::uno::XInterface** **XTextRange** **XSimpleText**	Je le savais déjà.
Methods' Summary	**createTextCursor** **createTextCursorByRange** **insertString** **insertControlCharacter**	Hourra !
Methods' Details	**createTextCursor** XTextCursor **createTextCursor();** **----------** **Returns** a new instance of a TextCursor service which can be used to travel in the given text context.	Donc MonTexte.createTextCursor (ligne 4 de la macro) est un curseur de texte (service TextCursor).
Derived Interfaces	**XText**	Tiens, dans XText on ne me parlait pas d'interface dérivée. Kesako ?
Services which Support this Interface	**...** **::com::sun::star::drawing::Text** **Text** **...**	Les mêmes que pour XText.

Après un petit détour par le Staroffice Programmer's Tutorial pour comprendre cette histoire d'interface dérivée, il me semble qu'on peut décrire la situation comme suit, en se limitant au module /com/sun/star/text/ :



Et voilà encore du vocabulaire :

Se lit :
Le service Text '*supports*' l'interface Text et l'interface XSimpleText.
L'interface XText est une '*exported interface*' du service Text.
Mais l'interface XSimpleText n'est pas une '*exported interface*' du service Text.

Se lit :
L'interface XText est une '*derived interface*' de l'interface XSimpleText.
ou
L'interface XText '*extends*' l'interface XSimpleText.
ou
L'interface XText est un client qui peut utiliser les méthodes de l'interface XSimpleText qui est donc un fournisseur. (cf. le " Staroffice Programmer's Tutorial ")

Cette expression " interface dérivée " m'a longtemps bloquée. Je la comprenais à l'envers. C'est pourquoi j"indique plusieurs mode de lecture, la version du " Staroffice Programmer's Tutorial " me paraissant plus claire. L'expression '*derived interface*' utilisée par l'API aurait tendance à me faire mettre les flèches dans l'autre sens, mais il paraît, dixit le " Staroffice Programmer's Tutorial ", que le sens des flèches est imposé par UML (Unified Modeling Language). Ne m'en demandez pas plus.

La méthode createTextCursor d'XSimpleText peut donc être utilisée pour un objet Text.

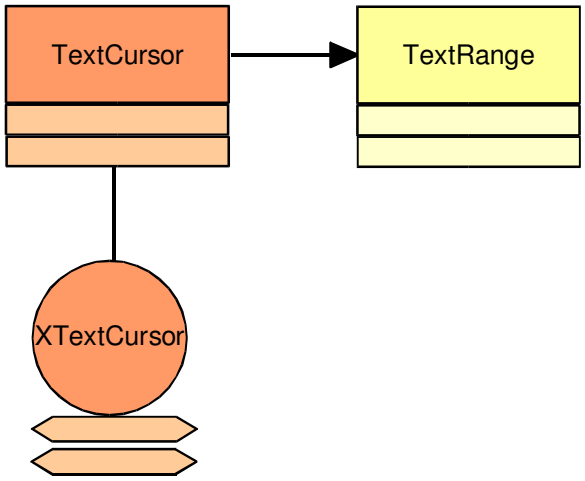
```
3  MonCurseur = MonTexte.CreateTextCursor
```

La ligne 3 est enfin expliquée. Ouf !
Evidemment il faut maintenant aller voir le service TextCursor puisque dans les lignes 5 à 7 apparaît 'MonCurseur.truc' . Et ça ressemble à l'appel d'une méthode d'interface liée à un service.

d) Service TextCursor

Service TextCursor		***Commentaire***
Description	**A TextCursor is a TextRange which can be moved within a :: com::sun::star::drawing::Text object.**	on dirait le jeu de l'oie : “ allez à la case textrange " ! Qu'est-ce que le module de dessin (*drawing*) vient faire là-dedans ?
Included Services	**TextRange**	Encore lui !
Exported Interfaces	**...** **XTextCursor** **...**	Je m'y attendais. Ça semble une habitude : service Machin, interface XMachin.
Services which Include this Service	**TextLayoutCursor** **TextViewCursor**	Pas intéressant. Comme pour le service XText, les services '*which include this service*' ne semblent pas être utiles.

Encore un petit détour par le Staroffice Programmer's Tutorial pour comprendre cette histoire de service inclus (*included service*) et voici un autre petit bout de schéma :



Et le jargon qui va avec :

Se lit :
Le service TextCursor inclut le service TextRange.
ou
Le service TextRange est un '*included service*' du service TextCursor.
ou
Le service TextCursor est (*is*) un service TextRange.
ou
Le service TextCursor est un sous-service du service TextRange.

Décidemment le sens de la flèche me gêne et j'ai là aussi tendance à comprendre l'expression '*included service*' à l'envers.

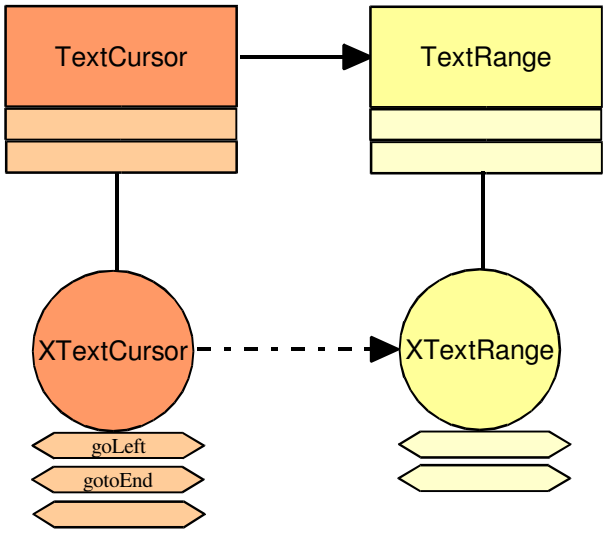
Et on ne peut pas dire que l'exploration du service TextCursor ait fait avancer le schmilblic !

Donc en route pour l'interface XTextCursor et le service TextRange. Espérons ...

e) Interface XTextCursor

Interface XTextCursor		Commentaire
Base Hierarchy	**::com::sun::star::uno::XInterface** **XTextRange** **XTextCursor**	
Methods' Summary	**...** **goLeft** **goRight** **gotoStart** **gotoEnd** **...**	Hé hé !
Methods' Details	**goLeft** boolean **goLeft**([in] short nCount,, **[in] boolean bExpand);**	C'est en accord avec la syntaxe de la ligne6.
Derived Interfaces	**...**	Pas intéressant.
Services which Support this Interface	**TextCursor**	OK.

Je complète le schéma précédent :



Les lignes 5 et 6 de la macro utilisent donc d'une manière standard les méthodes goLeft et gotoEnd en les appliquant à l'objet MonCurseur qui est un objet de type TextCursor.

```
5  MonCurseur.gotoEnd(false)
6  MonCurseur.goLeft(3,true)
```

Ça roule !

Il ne reste plus, je l'espère, qu'à étudier le service TextRange et surtout son interface XTextRange qui pourrait peut-être nous fournir des méthodes pour comprendre la ligne 7, à moins que l'API ne m'entraîne dans un nouveau jeu de piste.

f) Service TextRange

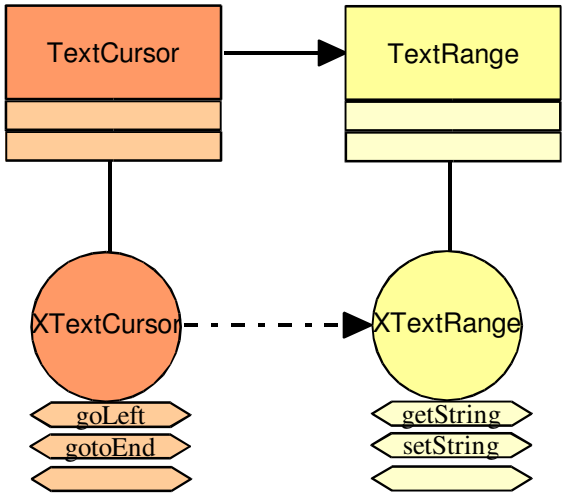
Service TextRange		*Commentaire*
Included Services	**...**	Rien d'intéressant.
Exported Interfaces	**...** **XTextRange** **...**	OK.
Services which Include this Service	**...** **TextCursor** **...**	Je le savais déjà.

Rien de nouveau.

g) Interface XTextRange

Interface XTextRange		*Commentaire*
Base Hierarchy	**::com::sun::star::uno::XInterface** **XTextRange**	Connu.
Description	Describes the object's position in a text. It represents a text range.	Heu ...
Methods' Summary	**getText** **getStart** **getEnd** **getString** **setString**	Youpi tra la la lère !
Derived Interfaces	**...** **XSimpleText** **XText** **XTextCursor** **....**	Connu.
Services which Support this Interface	**...** **Text** **TextCursor** **TextRange** **...**	Connu

On fignole le schéma :



Et la ligne 7 est élucidée :

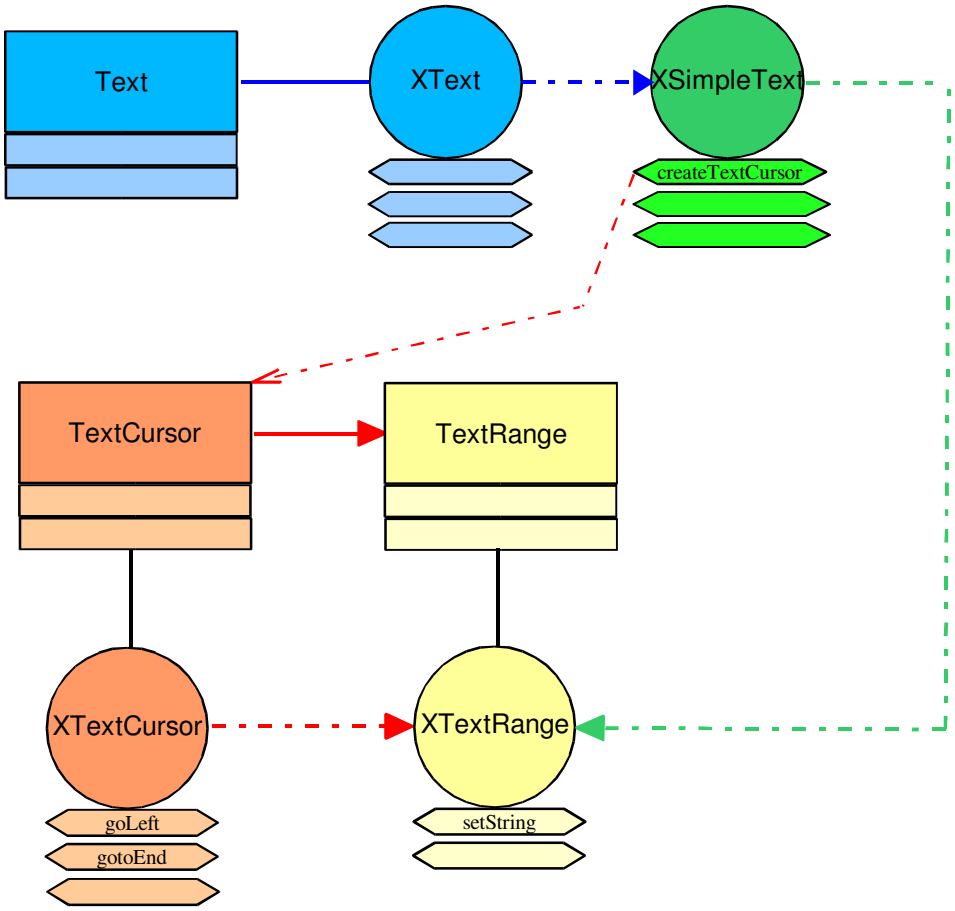
```
7  MonCurseur.String = "Fin"
```

est une version simplifiée de

```
7  MonCurseur.setString("Fin")
```

C'est fini !

2.4 Récapitulatif



J'ai rajouté une flèche pour rappeler qu'avec createTextCursor on crée un objet TextCursor.

Note : mon choix de flèche (pleine, pointillé etc.) est très libre et sans aucun rapport avec les conventions de l'UML tout simplement parce que je ne les comprends pas. Je ne me suis soumise qu'aux règles imposant le sens des flèches et c'est bien assez, vu que je ne m'y fais toujours pas.

Pour contourner ce problème de sens des flèches, j'ai tendance à traduire (version très libre) les flèches par " on a le droit d'utiliser ce qui est là [pointé par la flèche] " ou " avec cette méthode, je fabrique ça [pointé par la flèche] ". Alors la macro s'explique facilement : on a pris la piste bleue puis la piste rouge.

Rappel de la macro :

```
1  Dim MonDocument As Object, MonTexte As Object, MonCurseur As Object
2  MonDocument = ThisComponent
3  MonTexte = MonDocument.Text
4  MonCurseur = MonTexte.createTextCursor
5  MonCurseur.gotoEnd(false)
6  MonCurseur.goLeft(3,true)
7  MonCurseur.String = "Fin"  'équivalent de MonCurseur.getString("Fin")
```

Vous pouvez essayer de prendre la piste verte au lieu de la piste rouge, juste pour voir (supprimez les lignes 4, 5 et 6 et transformez la ligne 7 en MonTexte.String = "Fin"). Mais attention, avec un document d'essai, pas avec celui-ci , si vous y tenez.

Note : dans le " Developer's Guide " (partie du SDK) et dans le Staroffice Programmer's Tutorial, on trouve des schémas de ce type qui peuvent aider.

2.5 XRay, l'outil miracle

Même si vous m'avez suivie jusque là et même en supposant que vous y voyiez plus clair, je suis sûre que vous êtes très angoissé. Faudra-t-il faire ce parcours du combattant à chaque fois ?

Pas quand il s'agit de comprendre une macro car après tout, peu importe que vous sachiez que createTextCursor est une méthode de XSimpleText, interface dont dérive l'interface XText qui est elle-même une interface exportée du service Text (ouf !).
Tout ce que vous avez besoin de savoir, c'est que vous pouvez utiliser la méthode createTextCursor avec un objet texte. Et ça Bernard Marcelly vous l'avait dit dans son " API (presque) sans peine ".

Retour donc à la case départ ? A quoi cela a-t-il servi que je me décarcasse ?

C'est que Bernard ne nous a pas tout dit. Par exemple, il ne nous a pas dit comment aller à la fin d'une section. Et il nous a prévenu, pour aller plus loin il faut se plonger dans l'API. Bref, dès qu'on veut écrire une macro spécifique et qu'on ne trouve rien sur les sites de macros ou dans les documents, il faut savoir utiliser l'API.

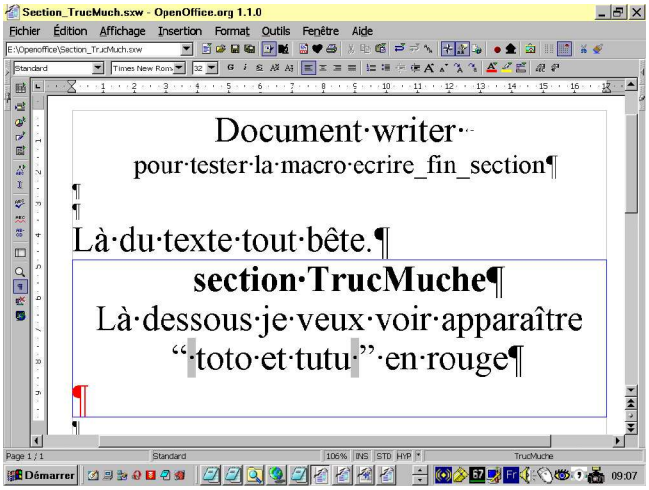
Mais heureusement Magic Bernard a conçu un outil, une macro encore, qui nous permet de savoir ce qu'on a le droit de faire avec un objet. C'est XRay, disponible à cette adresse http://fr.openoffice.org/Documentation/Index.html et qu'il faut télécharger de toute urgence. C'est tout simplement un outil indispensable, que j'utiliserai pour la deuxième partie, en combinaison avec l'API.

3 Ecrire une macro en utilisant l'API

3.1 Le but recherché

Comme indiqué précédemment il s'agit d'écrire du texte, en l'occurrence “ toto et tutu ", à la fin d'une section nommée “ TrucMuche ".
Note : “ toto et tutu " a peut-être une forte charge affective pour Tony Galmiche et je ne me permets pas de modifier ce texte :-)

Evidemment, pour fonctionner, la macro suppose qu'un fichier Writer comportant une section nommée " TrucMuche " soit ouvert :



3.2 L'analyse du problème

L'essentiel de la macro est déjà écrit : c'est celle que nous venons d'étudier, en l'adaptant, bien sûr. La seule difficulté nouvelle est de trouver comment on positionne un curseur à la fin d'une section. La macro devrait donc a priori ressembler à ça :

```
1  Dim MonDocument As Object, MonTexte As Object, MonCurseur As Object
2  MonDocument = ThisComponent
3  MonTexte = MonDocument.Text
4  MonCurseur = MonTexte.createTextCursor
5  'une ou plusieurs lignes à écrire pour trouver la section TrucMuche
6  'et placer le curseur à la fin de celle-ci
7  MonCurseur.String = "toto et tutu"
```

3.3 Point de départ pour la recherche de la solution

On dispose de deux outils : l'API et Xray.

L'API, dans le module com/sun/star/text mentionne un TextSection qui pourrait bien être un point de départ.

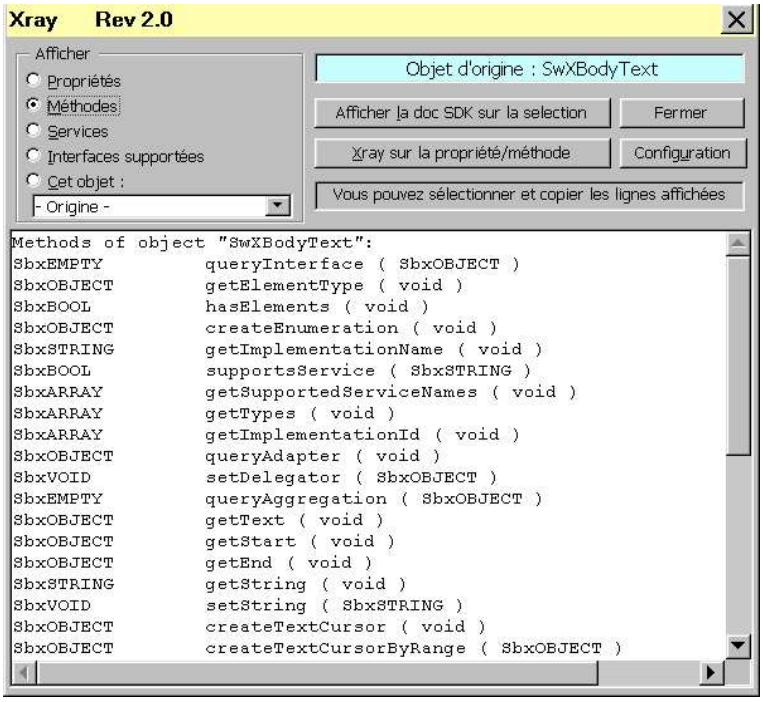
Avec XRay on peut regarder s'il n'y aurait pas quelque chose concernant les sections, accessibles à partir d'un objet texte.

Je connais trop bien le jeu de l'API : deux pas en avant, trois pas en arrière. Je choisis donc de démarrer avec XRay.

3.4 Construction de la macro

a) Recherche dans les méthodes de MonTexte (objet Text)

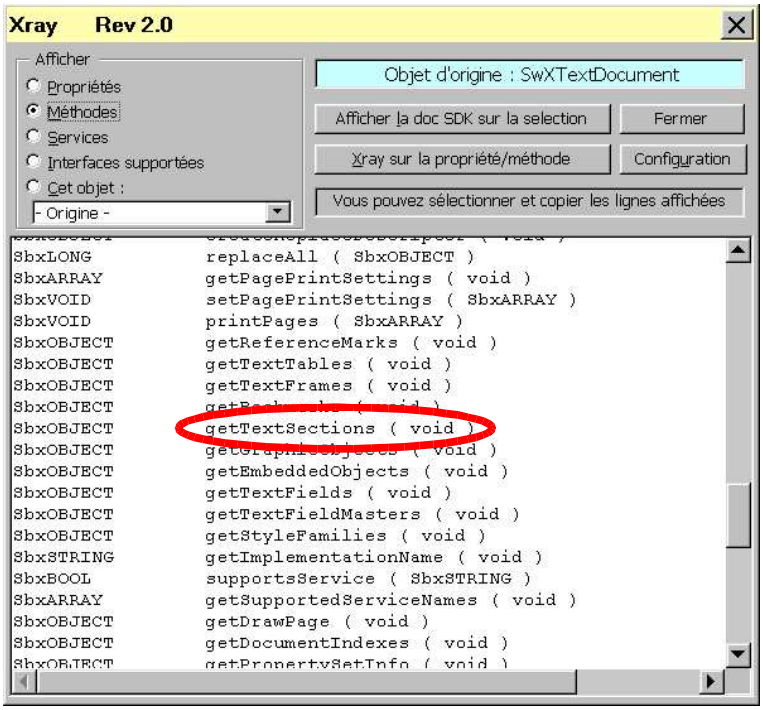
```
1  Dim MonDocument As Object, MonTexte As Object
2  MonDocument = ThisComponent
3  MonTexte = MonDocument.Text
4  XRay.XRay MonTexte
```



Ça commence mal : rien à se mettre sous la dent.
Pas envie d'aller voir l'API. Je vais remonter d'un cran pour explorer l'objet MonDocument avec XRay. Des fois que ...

b) Recherche dans les méthodes de MonDocument(objet TextDocument)

```
1  Dim MonDocument As Object, MonTexte As Object
2  MonDocument = ThisComponent
3  XRay.XRay MonDocument
4  MonTexte = MonDocument.Text
```



Bonne pioche ! Allez j'y vais :

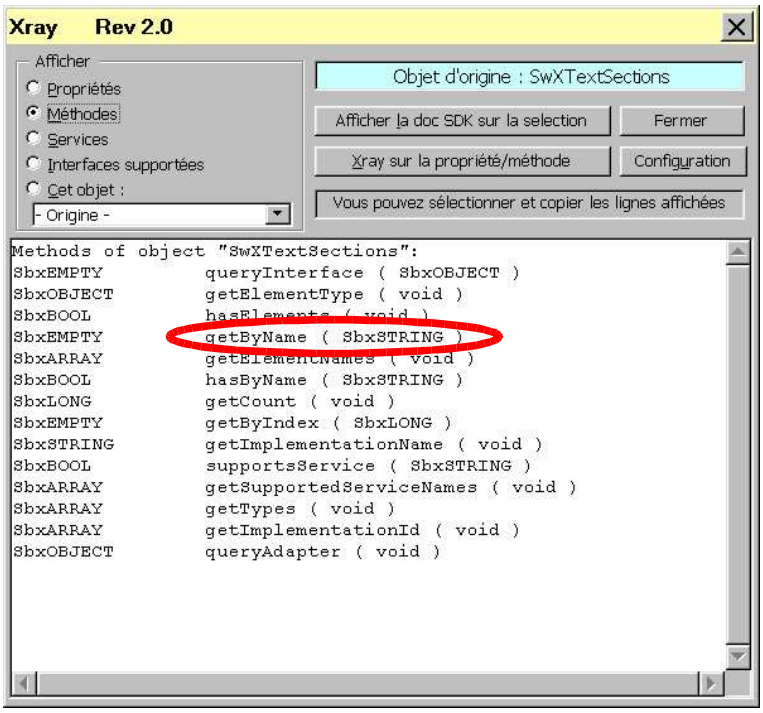
```
1  Dim MonDocument As Object, TextSections As Object
2  MonDocument = ThisComponent
3  TextSections = MonDocument.getTextSections
```

c) Recherche dans les méthodes de l'objet TextSections

Un petit coup de XRay

```
XRay.XRay TextSections
```

et en route :



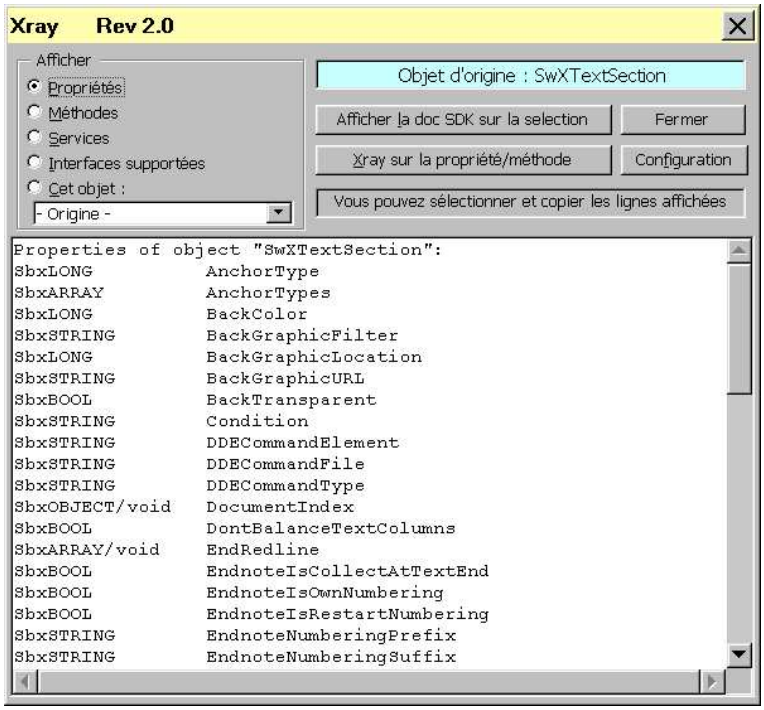
Je n'y crois pas ! C'est trop fastoche!

Je crée l'objet MaSection :

```
1  Dim MonDocument As Object,TextSections As Object
2  Dim MaSection As Object
3  MonDocument = ThisComponent
4  TextSections = MonDocument.getTextSections
5  MaSection = TextSections.getByName ("TrucMuche")
```

d) Recherche dans les méthodes de l'objet MaSection

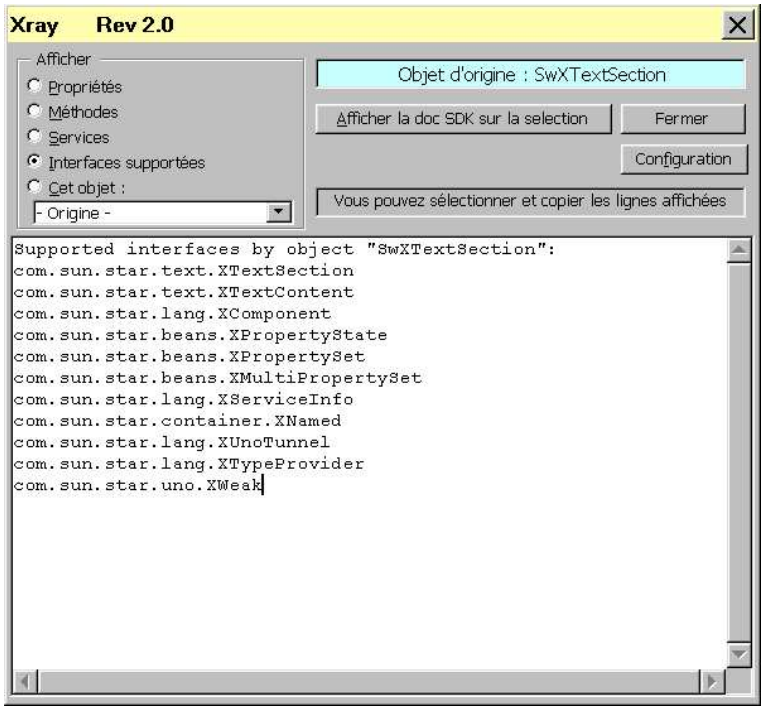
```
XRay.XRay MaSection
```



J'aurais aimé un gotoEnd mais rien en vue. Ç'aurait été trop beau.
Rien ne m'inspire. Là je suis plantée.

XRay me dit que MaSection est une ' *TextSection* ' (Objet d'origine : SwXTextSection)

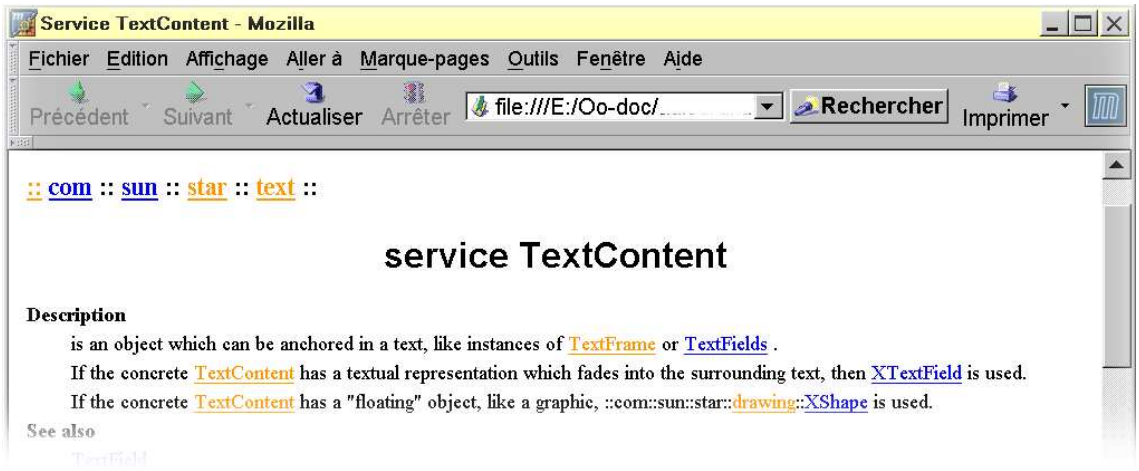
Je regarde les interfaces supportées : XTextSection (je m'en doutais) et XTextContent.



Un petit tour du côté de l'API s'impose.

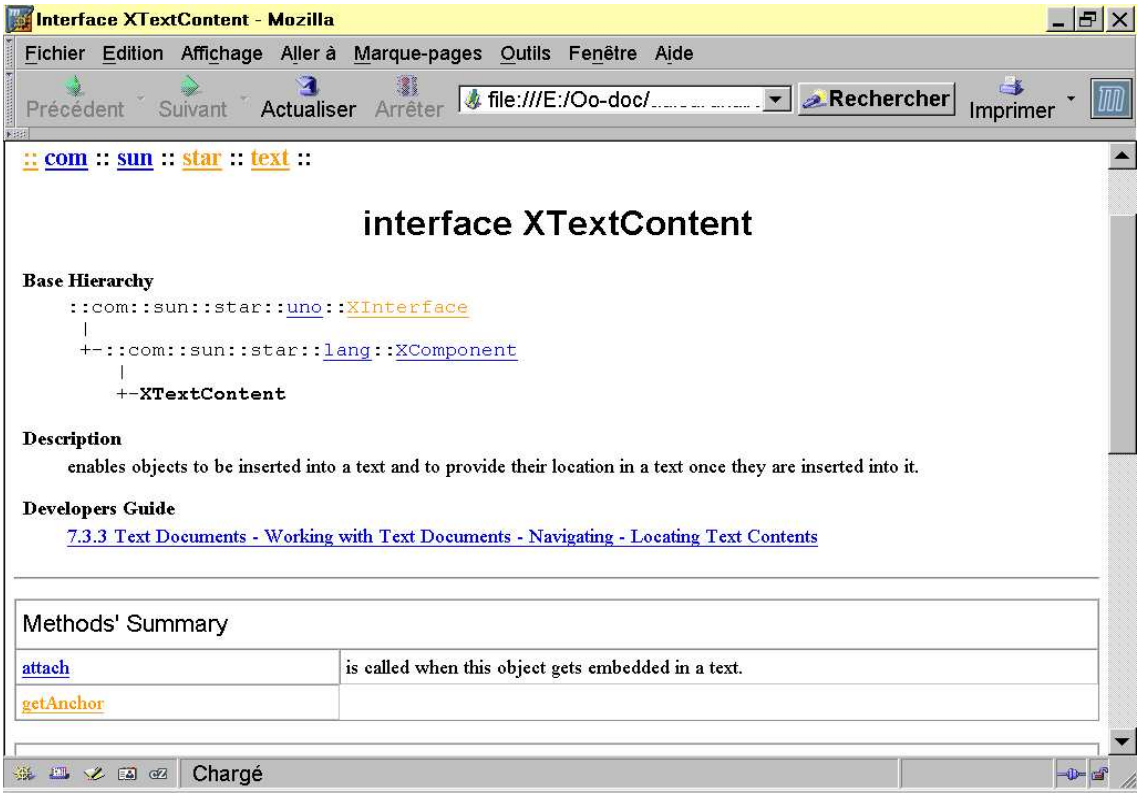
e) API : service TextContent et interface XTextContent

- Visite du serviceTextContent dans l'API



' *object which can be anchored in a text* ' :
Tiens j'avais vu un AnchorType, un AnchorTypes et un getAnchor dans les méthodes de MaSection. *Anchor* = ancre. Est-ce qu'une section est ancrée comme une image ou un cadre ? Jamais vu d'ancre, moi. Je laisse ma réflexion en suspens.

- Visite de l'interface XTextContent



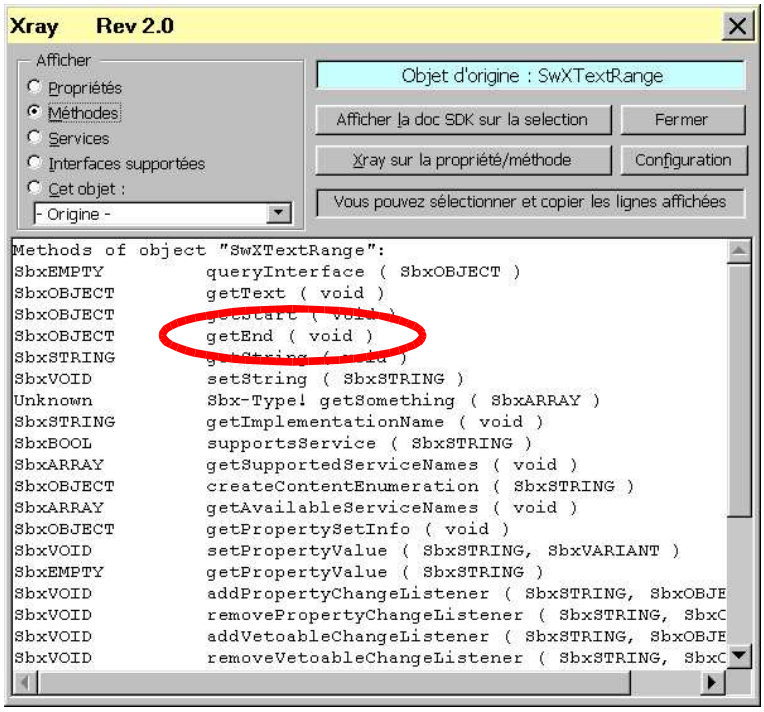
Donc ça permet d'insérer une section (ou autre chose sans doute) ou de savoir où elle se trouve si elle existe déjà. Piste intéressante. D'autant qu'une des méthodes est précisément getAnchor sur lequel je méditais il y a peu.

J'y vais !

```
1  Dim MonDocument As Object, TextSections As Object
2  Dim MaSection As Object, Anchor As Object
3  MonDocument = ThisComponent
4  TextSections = MonDocument.getTextSections
5  MaSection = TextSections.getByName ("TrucMuche")
6  Anchor = MaSection.getAnchor
```

f) Recherche dans les méthodes de l'objet Anchor

```
XRay.XRay Anchor
```



XRay m'indique que Anchor est un XTextRange.
Ah, celui-là je l'ai déjà vu dans la première partie (voir schéma récapitulatif)

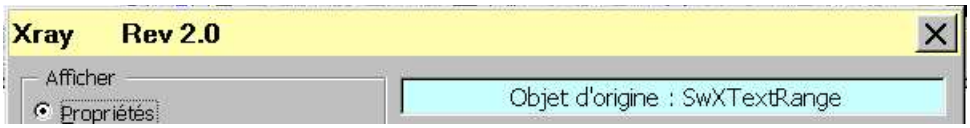
Et bingo, il existe un getEnd.
Je sens que je touche au but.

OK, j'attendais plutôt un gotoEnd mais bon, je sens que si je savais ce qu'est un TextRange je ne serais pas surprise d'avoir un getEnd. Mais j'en a marre de chercher, j'utilise tel quel :

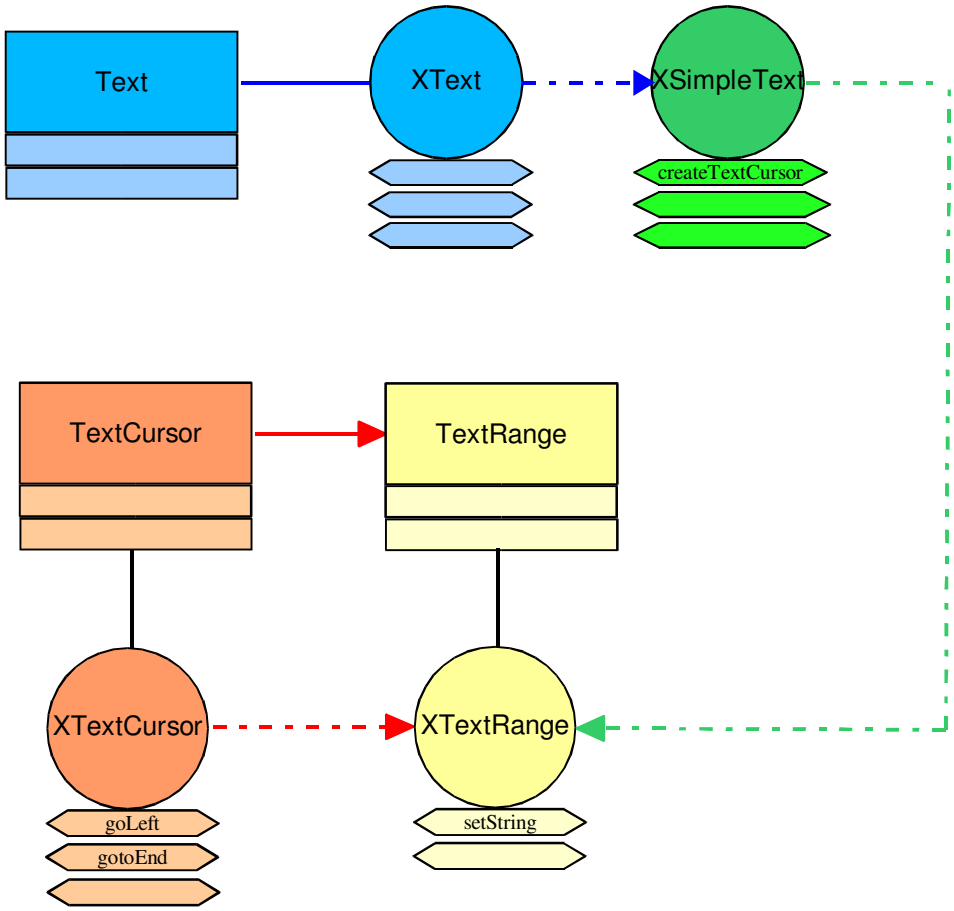
```
1  Dim MonDocument As Object, TextSections As Object
2  Dim MaSection As Object, Anchor As Object, FinSection As object
3  MonDocument = ThisComponent
4  TextSections = MonDocument.getTextSections
5  MaSection = TextSections.getByName ("TrucMuche")
6  Anchor = MaSection.getAnchor
7  FinSection = Anchor.getEnd
```

g) Recherche dans les méthodes de l'objet FinSection

```
XRay.XRay FinSection
```



FinSection est aussi un objet TextRange. Donc je risque de tourner en rond longtemps. Regardons d'un peu plus près mon schéma récapitulatif.



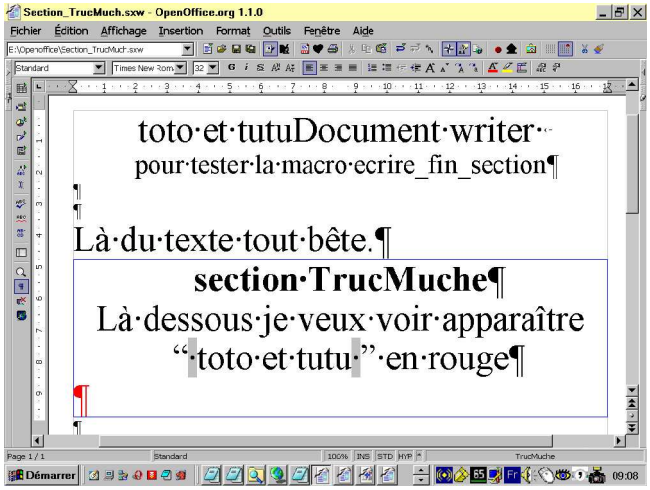
Bon sang, mais c'est bien sûr , pour accéder à un endroit précis d'un texte, il me faut un curseur. J'avais perdu de vue ce point là. En voiture, Simone !

```
1  Dim MonDocument As Object, MonTexte As Object,TextSections As Object
2  Dim MaSection As Object, Anchor As Object, , MonCurseur As Object,
3  MonDocument = ThisComponent
4  MonTexte = MonDocument.Text
5  TextSections = MonDocument.getTextSections
6  MaSection = TextSections.getByName ("TrucMuche")
7  Anchor = MaSection.getAnchor
8  FinSection = Anchor.getEnd
9  MonCurseur = MonTexte.createTextCursor
```

Et pendant que j'y suis, je rajoute

```
10 MonCurseur.setString("toto et tutu")
```

Olé !

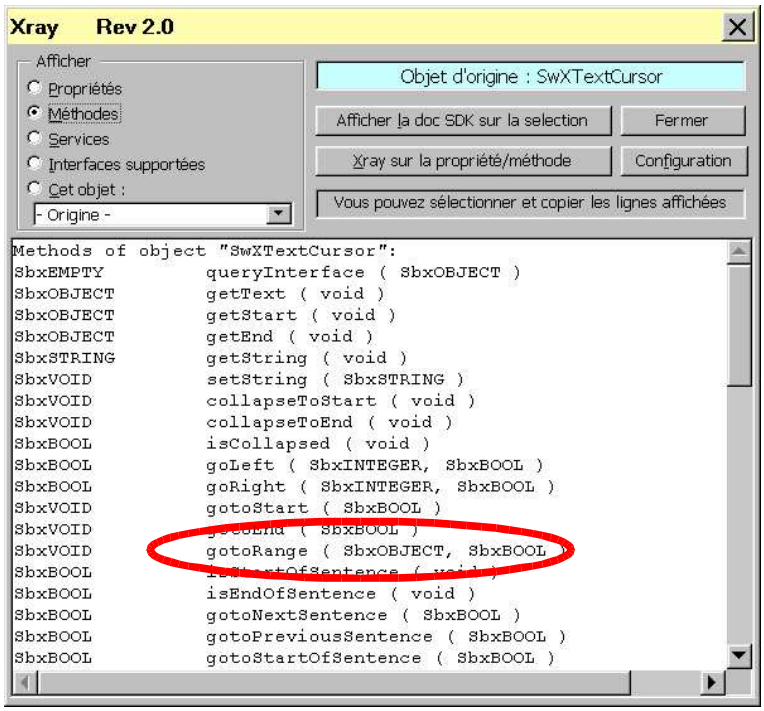


Euh ... Léger coup de blues là. Toto et tutu sont au début du texte et non pas en fin de section ...

Agnès	*Ma fille, ce n'est pas le moment de te décourager, y'a pas à tortiller, il faut te mettre au clair sur cette histoire de TextRange.*
Agnès (la fille)	*J'en ai marre de l'API, me sort par les oreilles celui-là !*
Agnès (la mère donc ?) compatissante, sans conviction	*Bon, on peut peut-être essayer de radiographier MonCurseur.*
Agnès (fille) *:*	*Ce sera toujours moins pire que de replonger dans l'API.*
[aparté *:*	*Chers amis, excusez cet instant d'égarement mais il se fait tard et vraiment, l'Agnès (mère-fille), l'API il lui sort aussi par les trous de nez et par ... tout.*]

Reprenons calmement.

h) Recherche dans les méthodes de l'objet MonCurseur



Reyoupie retra rela rela relère !

Le gotoRange(SbxObject, SbxBOOL), il doit vouloir dire Va_au_TextRange (QuiEstLà, BOOL). Pour le BOOL je ne sais pas trop, si ce n'est que j'ai le choix entre false et true. Je vais commencer par false, on verra bien.

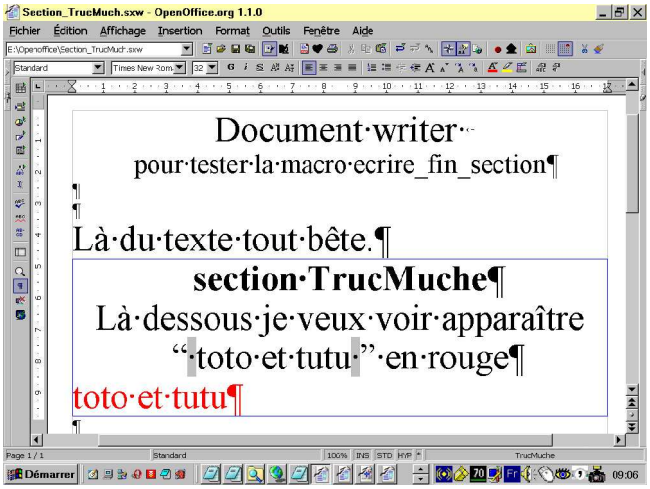
Ben oui, je n'avais pas dit au curseur de se positionner à FinSection. Si on ne lui dit pas, il ne le sait pas, le pauvre. Alors là, je concocte la ligne qui tue :

```
10  MonCurseur.gotoRange(FinSection, false)
```

ce qui donne :

```
1   Dim MonDocument As Object, MonTexte As Object, MonCurseur As Object,TextSections As Object
2   Dim MaSection As Object, Anchor As Object, FinSection As object
3   MonDocument = ThisComponent
4   MonTexte = MonDocument.Text
5   TextSections = MonDocument.getTextSections
6   MaSection = TextSections.getByName ("TrucMuche")
7   Anchor = MaSection.getAnchor
8   FinSection = Anchor.getEnd
9   MonCurseur = MonTexte.createTextCursor
10  MonCurseur.gotoRange(FinSection, false)
11  MonCurseur.setString("toto et tutu")
```

Toto et tutu sont enfin là où ils doivent être :



Et je vais là où je devrais être depuis longtemps : au lit !

3.5 Le lendemain : où l'on fait une beauté à la macro

J'en entends qui disent qu'il y a publicité mensongère. J'avais indiqué que la solution tenait en deux lignes et il y en a 5 de plus (eh, les Dim, ça compte pas !).

Qu'à cela ne tienne, il suffit de faire du 5 en 2

```
Dim MonDocument As Object, MonTexte As Object,TextSections As Object
Dim MaSection As Object, Anchor As Object, FinSection As object, MonCurseur As Object
MonDocument = ThisComponent
MonTexte = MonDocument.Text
MaSection = MonDocument.getTextSections.getByName ("TrucMuche")
MonCurseur = MonTexte.createTextCursor
MonCurseur.gotoRange(Masection.Anchor.getEnd, false)
MonCurseur.setString("toto et tutu")
```

On peut même encore gagner 2 lignes :

```
Dim MonDocument As Object, MonCurseur As Object
MonDocument = ThisComponent
MonCurseur = MonDocument.Text.createTextCursor
MonCurseur.gotoRange( MonDocument.TextSections.getByName ("TrucMuche").Anchor.End, false)
MonCurseur.setString("toto et tutu")
```

Ou en utilisant les pseudo-propriétés :

```
Dim MonDocument As Object, MonCurseur As Object
MonDocument = ThisComponent
MonCurseur = MonDocument.Text.createTextCursor
MonCurseur.gotoRange( MonDocument.TextSections.getByName ("TrucMuche").Anchor.End, false)
MonCurseur.String="toto et tutu"
```

4 Pour conclure

On a beau faire, l'API ce n'est pas simple, voire légèrement compliqué.

A mon avis, plutôt que d'aller se plonger dans l'API, autant y envoyer XRay le grouillot et piocher dans ce qu'il ramène, au flair. L'API, pour les apprentis programmeurs, c'est vraiment quand on est coincé, ou qu'on veut en savoir plus sur ce qu'il y a dans les filets de XRay.

Il y a encore mieux : profiter sans vergogne du travail des autres ! Rien de tel qu'une macro que l'on n'a plus qu'à adapter à son problème personnel. Mais pour ça il faudrait que chacun pense à mettre ses macros à disposition de tous.
On peut les envoyer à Sophie Gautier (sgauti@openoffice.org) qui les mettra sur le site http://fr.openoffice.org/
ou les déposer sur cet espace http://codesnippets.services.openoffice.org/ .
On peut même écrire des how-to regroupant des macros sur un thème précis : formulaires, sections,etc. et les transmettre à Sophie Gautier (sgauti@openoffice.org) .

Avis aux amateurs.

5 Crédits

Auteur : **Agnès Simonet**

Remerciement : **Tony Galmiche pour sa question qui a tout déclenché et ses critiques constructives, Bernard Marcelly pour son excellent how-to " L'API (presque) sans peine) " et tous les relecteurs .**

Intégré par : **Sophie Gautier**

Dernière modification : **7 juillet 2004**

Contacts : **Projet Documentation OpenOffice.org - Fr.OpenOffice.org**

Traduction :

6 Licence

Appendix

Public Documentation License Notice