DocBook

Table des matières

Qu'est que c'est ?

DocBook [http://www.docbook.org/] est un format XML destiné à la publication de livres et d'articles techniques. C'est d'ailleurs dans ce format que cet article à été rédigé :)

DocBook [http://www.docbook.org/] ne s'intéresse qu'au contenu du document et non à sa forme, ce seront les feuilles de style XSLT et CSS qui s'en chargeront. Il est ainsi possible avec un même document de le convertir en de nombreux formats : HTML, XHTML, PDf, RTF, Latex ...

Il est très simple d'en faire du versionning puisque c'est du XML :)

Prérequis:

Installer ces quelques outils:

```
apt-get install docbook docbook-xml docbook-xsl xlstproc libxml2-utils fop enscrip
```

Et pour insérer des images au PDF il faudra installer la library Java : http://java.sun.com/products/jimi/

```
tar xvzf jimi1_0.tar.Z
cp Jimi/JimiProClasses.zip /usr/share/java/jimi-1.0.jar
```

Validité du document

Pour qu'un document DocBook [http://www.docbook.org/] puisse êtr exploité il doit être **valide**

C'est à dire qu'il doit respecter les standards de DocBook ou plus exactement être conforme à la DTD [http://fr.wikipedia.org/wiki/DTD] de docbook

De nombreux outils sont disponible pour vérifier la validité de documents. J'utilise dans cet exemple **xmllint** appartenant au package libxml2-utils

```
xmllint --valid test.xml
```

Pour que celui-ci puisse vérifier la conformité du document il est necessaire qu'il sache à quelle DTD il à a faire. C'est dans le document Docbook que nous lui préciserons :

```
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
```

Création d'un document Docbook

Débutons tout de suite la rédaction d'un document DocBook.(`test.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.4//EN"
"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd">
<article lang="fr">
      <title>Le titre de mon article</title>


      <section>
            <title>Le titre de la section</title>

            <para>Ceci est un paragraphe ...</para>

            <para>Ceci en est un autre ...<emphasis>Simple non ?</emphasis></para>

     </section>

</article>
```

Détaillons les balises mises en scène.

La première <?xml ... ?> indique que notre document sera de type Article', la version de XML utilisée (1.0 ou 1.1) et le codage des caractères.

La ligne <!DOCTYPE ...> précise la DTD utilisée.(Celle qui nous permet de vérifier la validité du document)

Et enfin le corps du document. Il existe environ 400 balises sous DocBook je ne vais pas m'éterniser sur celles-ci. Pour plus d'infos voir http://www.docbook.org/tdg/

Transformation du document Docbook

La transformation est produite par des feuilles de style XSLT fournie dans le package **docbook-xsl.** Avec celles-ci il nous est déjà possible de convertir notre document en html, xhtml, manpage, fo et worldml dont je ne connais pas l'utilité.

Pour appliquer l'une de ces feuilles de styles il nous faut un outil : **xsltproc** est très rapide pour celà.

Testons de suite :

```
xsltproc --stringparam LANG fr \
         --stringparam section.autolabel 1 \
         --stringparam toc.section.depth 3 --stringparam generate.toc.level 2 \
         --stringparam html.stylesheet  mafeuille.css /usr/share/sgml/docbook/styl
         test.xml > test.xhtml
```

On constate que l'on peut passer des paramètres à xsltproc pour affiner la transformation. Ainsi la commande ci-dessus nous produiera un document XHTML avec une table des matières et des chapitre numérotés.

Pour une trnasformation en PDF il nous faut passer par une étape/outil supplémentaire: FOP http://xmlgraphics.apache.org/fop/

Tout dabord transformation en 'fo' (ne pas oublier l'installation de la librairie Jimi [http://www.xmlmind.com/xmleditor/namespace/clipboard" > <ulink url="http://java.sun.com/products/jimi/" >http://java.sun.com/products/jimi/])

```
xsltproc --stringparam LANG fr --nonet -o test.fo /usr/share/sgml/docbook/styleshe
```

et enfin en PDF

```
fop -fo test.fo test.pdf
```

Utilisation de <programlisting>

'programlisting' permet comme son nom l'indique de baliser le code source de programmes. Justement ce dont j'avais besoin :)

```
<programlisting lang="perl">#!/usr/bin/perl

$a="toto";

print $a;</programlisting>
```

J'intègre ce bout de docbook à `test.xml` et le transforme en XHTML :

```
<pre class="programlisting">#!/usr/bin/perl
$a="toto";
print $a;</pre>
```

Mmmm c'est ça le programlisting :(

Les feuilles de styles xslt de Docbook ne permettent pas encore de coloriser directement le code source, dommage :(

Qu'a ce ne tienne, on va s'y prendre autrement, il existe de nombreux outil permettant la colorisation de code source mais je n'en ai pas trouvé qui colorisent le code et seulement lui, de notre `test.xhtml`

Par contre enscript colorise à merveille de nombreux langages. Par exemple :

```
enscript -j -w html -Eperl --color -o out.html test.pl
```

Il serait intéressant de pouvoir utiliser ce programme sur les seules portions de code de notre fichier

`test.xhml`. Mais Arghh malheur Docbook lors de la transformation a supprilmer la notion de langage dans le code xhtml.

On pourrais peut être la rajouter sous une autre forme avant la conversion en xhtml ? Certes il nous est aussi possible de modifier les feuilles XSLT mais là je ne le sens pas :). Si quequ'un connais ?

Les étapes de transformation se présenterons ainsi:

- Ecriure du document Docbook avec les <programlisting lang="...">
- Ajout automatique d'un marquage pour chaque portion programlisting du fichier Docbook
- Transformation en XHTML
- Colorisation du code selon les marquages

Les prochains chapitres décrivent cette transformation en commençant par le choix de l'éditeur de document DocBook.

Choix d'un éditeur DocBook

Encore une fois plusieurs outils sont disponibles sur le Net pour effectuer cette tâche. Les indispensables 'vim' et 'emacs' diposent tous deux d'un mode XML qui permet l'indentation,la colorisation des balises, la complétion des balises mais comme vous l'aurez compris on écrit en mode 'balises'. Mais ça serait sympa de disposer d'un éditeur DocBook wysiwyg. Etrange me direz vous, le but de DocBook étant justement de séparer le fond de la forme alors pourquoi un éditeur wysiwyg ?. Le rêve ce serait d'avoir un outil qui nous propose, selon le positionnement dans le document, les divers balises utilisables. J'en ai testé plusieurs ...

J'ai tout naturellement cherché à savoir ce qui pouvait se faire avec OpenOffice. Sur http://wiki.docbook.org/topic/OpenOffice on trouve divers manière de travailler avec DocBook et notamment avec l'utilisation d'un template spécifique.Malheureusement il ne permet que le DocBook simplifié [http://www.docbook.org/xml/simple/] et encore :(. Jaxe [http://jaxe.sourceforge.net/Jaxe.html] de même ne permet que le DocBook simplifié [http://www.docbook.org/xml/simple/] :(

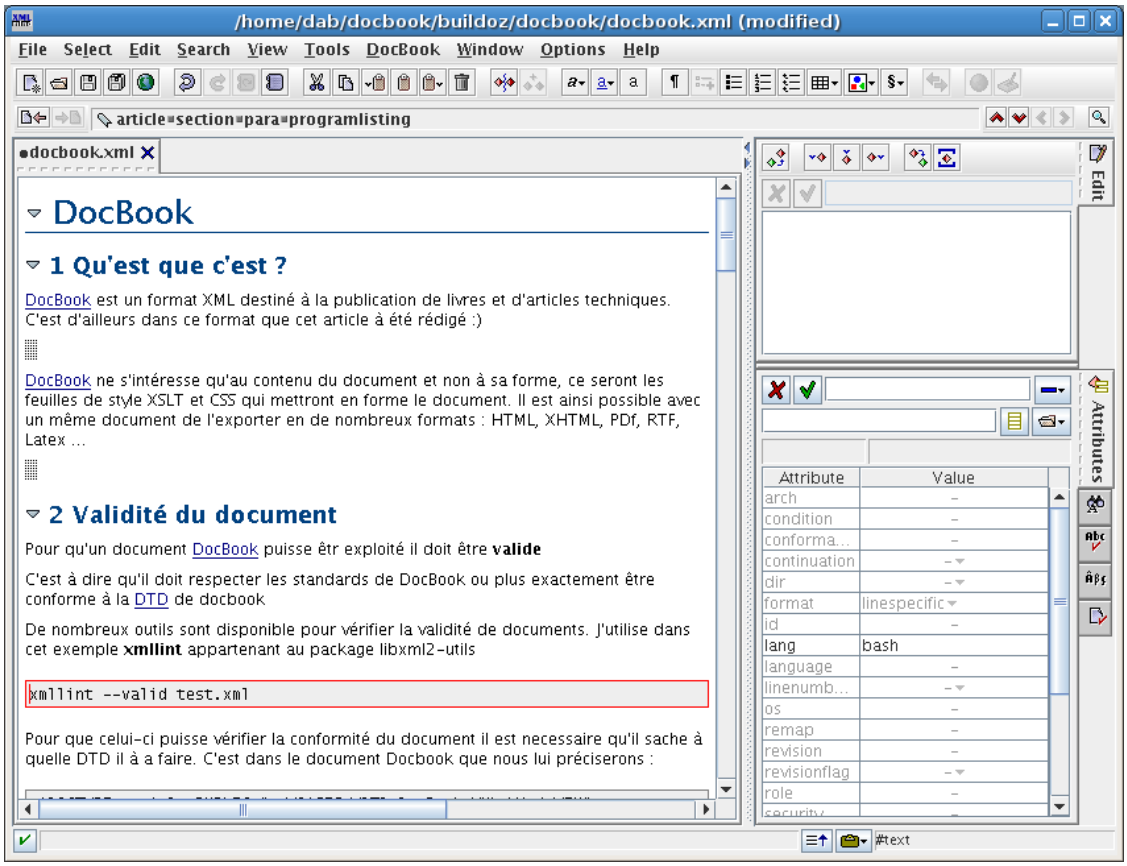
J'ai ensuite testé Conglomerate [http://www.conglomerate.org/] dont l'objectif est de devenir l'editeur XML que chacun peut utiliser.(In particular, our primary goal is to create the ultimate editor for DocBook and similar formats.). Mouai ... surement, mais pour l'instant il plante :(du moins avec Debian Testing.

Si vous connaissez d'autres produits opensource qui répondent à mes attentes je suis preneur.

Du coup je me suis rabattu sur un outil commercial mais dont on peut télécharger une 'Standard Edition' il s'agit de XXE [http://www.xmlmind.com/xmleditor/].

Simple d'utilisation il me va très bien. Toutefois cette version ne permet pas la conversion en PDF... peu importe je veux seulement qu'il me produise du DocBook je ferais moi même les conversions ;)

Voilà a quoi ça ressemble lors de l'édtion de cet article :)



Sur cette copie d'écran le curseur est positionné sur la balise 'programlisting', vous aurez remarqué que l'attribut 'lang' nous renseigne sur la nature du code source.

Lors du prochain chapitre je vais justement recherché ce fameux attribut 'lang' des noeuds 'programlisting' pour y ajouter un marqueur.

Vex [http://vex.sourceforge.net/] un plugin pour Eclipse permet aussi l'édition de document DocBook. (pas testé)

Promenade dans un document DocBook

Le format DocBook étant du XML, il existe de nombreuses manières de le triturer. Je souhaites ajouter à l'interieur des balise 'programlisting' le marqueur suivant:

```
[ lang=perl]
< ![CDATA[
...
 ] ]>
[ /lang]
```

La balise CDATA permet quant à elle de préserver les caratères spéciaux tels que <,>,&, ...

Dans ce chapitre j'utiliserai l'inteface SAX connu pour ces performances lorsqu'il s'agit de lire tous les

noeuds d'un document XML.

Linux Magazine dans son numéro 26 de Mars 2001 publie justement un article intitulé 'Introduction aux interfaces SAX et DOM'. Dans ces pages sont détaillées l'analyse de document XML. Les sources qui suivent sont largement inspirées de cet article.

Ah j'oubliai, le langage utilisé est Perl :)

Notre script se nomme `addMark2dbk.pl` :

```
#!/usr/bin/perl

# En entree: un fichier docbook
#
# ------------------------------

use strict;
use XML::Parser::PerlSAX;
use myHandler;

my $fichier = shift || die "Aucun fichier a analyser ! \n";
print STDERR " FICHIER=$fichier\n";

my $my_Handler = myHandler->new;

# Creation du parser
my $parser = XML::Parser::PerlSAX->new;

my @dochl=$parser->parse(
                Source => { SystemId => $fichier } ,
                Handler => $my_Handler ) ;

binmode(STDOUT,':utf8');
print @dochl;

# ------------------------------
```

Ce script ne fait que 'parser' le document fourni en entrée et lorsque apparait un 'événement' il fait appel au Handler.

Les événement sont de 3 types :

- Ouverture/Fermeture du document : on fait appel aux méthodes start/end_document
- Ouverture/Fermeture d'élément : on fait appel aux méthodes start/end_element
- Contenu de la balise : on fait appel à la méthode characters

Et donc notre handler (`myHandler.pm`) se présente comme ceci:

```
package myHandler;

use strict;
use vars qw ( $AUTOLOAD );

my $hl_node="programlisting";
my $hl_attrib="lang";
my $indent="    ";
my $nindent=0;
```

```
my $marge;
my $colorize=0; my $tocolorize;
my $lang;
my $doc;

sub new {
        my $type = shift;
        my $self = ( $#_ == 0 ) ? shift : { @_ };
        return bless $self, $type;
}

sub start_document{
        $doc .= "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
        $doc .= "<!DOCTYPE article PUBLIC \"-//OASIS//DTD DocBook XML V4.4//EN\"\n
        $doc .= "\"http://www.oasis-open.org/docbook/xml/4.4/docbookx.dtd\">\n";}

sub end_document{
        #$doc .= "[Fin de l'analyse du document]\n";
        return $doc;
}

sub start_element{
        my $xp = shift;
        my $el = shift;

        $marge = ${indent}x${nindent};
        my @Attributes = keys %{$el->{Attributes}};
        my $name = $el->{Name};
        $doc .= "$marge<$name";
        foreach my $att (@Attributes){
                my $val = $el->{Attributes}->{$att};
                $doc .= " $att=\"$val\"";
                if (( $att eq $hl_attrib )&&($el->{Name} eq $hl_node )){
                        $lang=$el->{Attributes}->{$att};
                        $colorize=1;
                }
        }
        $doc .= ">";
        $nindent++;
}

sub end_element{
        my $xp = shift;
        my $el = shift;

        $nindent--;
        my $name = $el->{Name};
        if ( $el->{Name} =~ /$hl_node/ ){

                $tocolorize =~ s/&gt;/>/g;
                $tocolorize =~ s/&lt;/</g;

                $doc .= "\n\[lang =$lang\]\n<![CDATA[$tocolorize] ] >\n\[\/lang\]\

                $colorize=0;
                $lang="";
                $tocolorize="";
        }
        $doc .= "</$name>\n";

}
sub characters{
        my $xp = shift;
        my $el = shift;
```

```
        my @keysel=keys %$el;

        if (  $colorize ne "0" ){
                $tocolorize .= $el->{Data};

        }
        else{
                $doc .= $el->{Data} if ( defined $el->{Data} );
        }
}
1;
```

En utilisant ce programme avec `addMar2dbk.pl` en entrée on obtient:

```
<programlisting lang ="perl">
[lang =perl]
< ![CDATA[#!/usr/bin/perl

# En entree: un fichier docbook
#
# ------------------------------

use strict;
use XML::Parser::PerlSAX; use myHandler;

my $fichier = shift || die "Aucun fichier a analyser ! \n";
print STDERR " FICHIER=$fichier\n";

my $my_Handler = myHandler->new;

# Creation du parser my $parser = XML::Parser::PerlSAX->new;

my @dochl=$parser->parse(
                Source => { SystemId => $fichier } ,
                Handler => $my_Handler ) ;

binmode(STDOUT,':utf8');
print @dochl;

# ------------------------------
] ]>
[/lang]
</programlisting>
```

OK mon marquage est bien effectué :)

Il ne me reste plus qu'a parser le fichier marqué, extraire ce qu'il y a entre [lang =..] et [/ lang] et le fournir à un programme de colorisation de code source. C'est ce que nous voyons dans le prochain chapitre.

Colorisation du code

J'ai choisi '**enscript**' le premier programme de colorisation qui m'est tombé sous la main mais d'autres choix sont possibles.

```
enscript --help-highlight
```

Cette commande permet de lister les langages connus par **enscript**. soit une soixantaine. Par contre je viens de m'apercevoir que le php n'est pas dans la liste. Et donc a remplacer plus tard.

Le programme Perl suivant parse le fichier 'marqué' et colorize son code. (`colorize.pl`)

```
#!/usr/bin/perl

use strict;

my $fichier = shift || die "Aucun fichier a coloriser !";

open(F,$fichier) or die "Lecture impossible de '$fichier' : !$\n";
my @F=<F>;
close F;

my $lang;
my $colorize=0;
my $tocolorize;
my $doc;

foreach my $l (@F){

        if ( $l =~ /\[lang =(.*)\]/ ){
                $lang=$1;
                $colorize=1;
        }
        elsif ( $l =~ /\[\/lang\]/ ){

                $tocolorize =~ s/&lt;/</g;
                $tocolorize =~ s/&gt;/>/g;
                $tocolorize =~ s/&amp;/&/g;

                # Pour faire un cat plutot qu'un echo qui lui
                # interprete les variables, ", ' ...
                open(TMP,">/tmp/tocolorize.tmp");
                print TMP $tocolorize;
                close TMP;

                my $srccolor = `cat /tmp/tocolorize.tmp | enscript -j -w html -E$l

                # Suppression du premier retour chariot
                $srccolor = substr($srccolor,1,-1);
                $doc .= $srccolor;

                # Raz des variables
                $colorize=0;
                $lang="";
                $tocolorize="";
        }
        else{
                if ( $colorize ){
                        $tocolorize .= $l;
                }
                else{ $doc .= $l }
        }
}

print $doc;
```

Eviter le copier/coller des sources utiliser plutôt le fichier contenant toutes les sources de cet article, car j'ai dû légèrement modifier le contenu du 'progmlisting' qui lui même contient les marqueurs. Et donc à éviter.

Les sources de l'article

En guise de conclusion j'ai fait un Makefile qui automatise ce ce qui a été dit.

L'archive ColorizeDbk.tgz [http://dab.free.fr/files/docbook/ColorizeDbk.tgz] contient les fichiers suivants:

```
find ColorizeDbk/
ColorizeDbk/
ColorizeDbk/Makefile
ColorizeDbk/addMark2dbk.pl
ColorizeDbk/colorize.pl
ColorizeDbk/docbook.xml
ColorizeDbk/myHandler.pm
ColorizeDbk/feuille.css
```

Un simple make Créera les versions xhtml et pdf du document `docbook.xml` :)