Schéma de données adapté, avec Docbook

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 mai 2009

https://www.bortzmeyer.org/docbook-custom-schema.html

Le format Docbook est un des plus utilisés pour la documentation technique, notamment dans la secteur informatique. La très grande majorité des grands logiciels libres utilisent Docbook. Ce format est très riche, avec plusieurs centaines d'éléments possibles. Malgré cette richesse, on a parfois besoin d'ajouter des élements ou bien d'effectuer d'autres adaptations locales à une organisation ou un projet. Comment faire avec Docbook ?

Les avantages de Docbook sont évidents : format ouvert <https://www.bortzmeyer.org/formats-ouverts.html>, indépendant de tout vendeur, exprimé sous forme de schéma XML pour profiter des outils XML existants et pouvoir être traité par des outils divers comme les VCS (contrairement aux formats binaires), Docbook s'est largement imposé. Mais chaque projet ou chaque organisation ressent parfois le besoin d'**adapter** le format standard et cet article expose la façon de le faire, pour le Docbook fondé sur les DTD (jusqu'à la version 4 incluse ; ensuite, à partir de la version 5 <http://www.docbook.org/schemas/5x> - publiée en 2008, Docbook utilise un langage de schéma bien plus perfectionné, RelaxNG, qui change nettement les choses).

Bien sûr, Docbook est un format ouvert, on peut tout simplement prendre les sources du schéma et les éditer. Mais, en procédant ainsi, on sort vraiment de Docbook et on aura du mal, par exemple, à intégrer les modifications futures. Il vaut donc mieux étendre Docbook de la manière « prévue pour ».

Pour prendre un exemple concret, on veut citer souvent, dans son document écrit en Docbook, les RFC. XML n'offrant hélas pas de macros <https://www.bortzmeyer.org/macros-en-xml.html>, la méthode correcte est d'ajouter un **élément** <rfc>. Pour cela, on écrit une DTD qui définit cet élément :

```
<!ELEMENT rfc EMPTY>
<!ATTLIST rfc num CDATA #IMPLIED>
```

Ici, l'élément est défini comme n'ayant pas de contenu (`EMPTY`) et prenant un attribut obligatoire, `num`, le numéro du RFC. Cela permet d'écrire `<rfc num="5514"/>` (mais pas `<rfc>IPv6 over social networks</rfc>`, puisque dans ce cas, `num` manque et le contenu n'est pas vide).

Il reste à dire à Docbook que cet élément est acceptable et où on peut le trouver. Pour faciliter cela, Docbook est réalisé de manière modulaire et beaucoup de termes DTD ont été définis comme vide, leur définition permettant de les remplir éventuellement avec des éléments locaux. C'est le cas du terme `local.para.char.mix` qui indique quels éléments supplémentaires peuvent apparaître dans un paragraphe (`<para>`). Par défaut, il est vide, remplissons-le avec notre nouvel élément :

```
<!ENTITY % local.para.char.mix
        "|rfc">
```

Désormais, comme `local.para.char.mix` est ajouté à la définition des éléments acceptables dans un `<para>`, `<rfc>` est acceptable (la barre verticale signifie OU BIEN).

Le document doit ensuite indiquer qu'il utilise, non pas le Docbook standard mais la version étendue. Si l'en-tête avec la version standard ressemble à :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
                  "dtd/xml/4.3/docbookx.dtd">
<article>...
```

celui d'un document utilisant notre Docbook adapté à nos besoins sera :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN" "dtd/xml/4.3/docbookx.dtd"[
<!ENTITY % dtd_custom SYSTEM "myschema.dtd">
%dtd_custom;
]>
<article>...
```

Maintenant, on peut écrire des documents complets correspondant au schéma « amélioré » par exemple :

```
<article>
<para>Ceci est un exemple de document <ulink
url="http://www.docbook.org"> Docbook</ulink> avec une référence à un
<rfc num="5241"/> (<foreignphrase>Naming Rights in IETF
Protocols</foreignphrase>).</para>
</article>
```

et ils sont validables (ici avec xmllint <http://xmlsoft.org>) :

```
% xmllint --noout --valid example.db
%
```

Maintenant, comment les traiter, par exemple pour une transformation en HTML afin de les servir sur le Web ? Essayons d'abord avec XSLT. Les programmes standard ne connaissent évidemment que les éléments officiels de Docbook et pas notre addition. Mais XSLT étant modulaire, il est très facile de rajouter un traitement pour nous :

```
<xsl:template match="rfc">
    <xsl:variable name="href">
      <xsl:text>http://www.ietf.org/rfc/rfc</xsl:text><xsl:value-of select="@num"/><xsl:text>.txt</xsl:text>
    </xsl:variable>
     <a href="{$href}"><xsl:text>RFC </xsl:text><xsl:value-of
        select="@num"/><xsl:text></xsl:text></a><xsl:if
        test="not(preceding::rfc)"> (RFC signifie Request For
        Comments. Ce sont les textes fondamentaux de
        l'Internet. Toutes les normes Internet sont des RFC, l'inverse
        n'est pas forcément vrai. Tous les RFC sont
        librement disponibles en ligne sur le <a
        href="http://www.ietf.org/">serveur de l'IETF</a>, l'organisme
        qui les édite. Plus de détails sont accessibles sur le <a
        href="http://www.rfc-editor.org/">serveur de l'éditeur des RFC</a>.)</xsl:if>
</xsl:template>
```

Ce gabarit XSLT traite tout élément nommé <rfc> et produit de l'HTML avec un lien vers le serveur Web de l'IETF et, si l'élément <rfc> est le premier du document (not(preceding::rfc)), un petit texte explicatif.

Pour les traditionnalistes qui préfèrent DSSSL <https://www.bortzmeyer.org/dsssl.html>, une version très simplifiée du code XSLT ci-dessus (elle est prévue pour l'impression uniquement, sans lien hypertexte) est :

```
(element rfc
  (let (
        (num (attribute-string "num"))
        )
    (if num
          (literal (string-append
                    "RFC "
                    num
                    ))
          (literal "RFC UNKNOWN")
    )))
```

Un exemple plus complexe vient lorsqu'on essaie d'utiliser XInclude avec Docbook, par exemple pour inclure une bibliographie <https://www.bortzmeyer.org/bibliographie-docbook.html>. La DTD complète pour faire cela est :

---

https://www.bortzmeyer.org/docbook-custom-schema.html

```
<!-- http://www.sagehill.net/docbookxsl/ModularDoc.html#UsingXinclude -->

<!ELEMENT xi:include (xi:fallback?) >
<!ATTLIST xi:include
    xmlns:xi   CDATA       #FIXED    "http://www.w3.org/2001/XInclude"
    href       CDATA       #REQUIRED
    parse      (xml|text)  "xml"
    encoding   CDATA       #IMPLIED >

<!ELEMENT xi:fallback ANY>
<!ATTLIST xi:fallback
    xmlns:xi   CDATA   #FIXED   "http://www.w3.org/2001/XInclude" >

<!-- Where are they allowed? -->

<!ENTITY % local.divcomponent.mix "| xi:include">   <!-- inside chapter or section elements
<!ENTITY % local.para.char.mix "| xi:include">      <!-- inside para, programlisting, literallayout, etc.
<!ENTITY % local.info.class "| xi:include">         <!-- inside bookinfo, chapterinfo, etc.

<!ATTLIST book
       xmlns:xi   CDATA       #FIXED    "http://www.w3.org/2001/XInclude">
<!ATTLIST article
       xmlns:xi   CDATA       #FIXED    "http://www.w3.org/2001/XInclude">
<!ATTLIST chapter
       xmlns:xi   CDATA       #FIXED    "http://www.w3.org/2001/XInclude">
<!ATTLIST section
       xmlns:xi   CDATA       #FIXED    "http://www.w3.org/2001/XInclude">
```

Et un éditeur sensible à la syntaxe comme psgml <http://www.lysator.liu.se/projects/about_psgml.html> peut alors éditer le document Docbook contenant les <xi:include> sans problème.

La documentation de référence sur les adaptations locales de Docbook est *"DocBook : The Definitive Guide"* <http://docbook.org/tdg/en/html/ch05.html>.