# Introduction au Projet de Documentation de FreeBSD pour les nouveaux participants

**Nik Clayton**

# Introduction au Projet de Documentation de FreeBSD pour les nouveaux participants

Nik Clayton
Version: 43184


# Résumé

Merci de votre participation au Projet de Documentation de FreeBSD. Votre contribution est très utile.

Cette introduction décrit tout ce que vous devez savoir pour commencer à participer au projet de documentation de FreeBSD, des outils et logiciels que vous utiliserez (indispensables et facultatifs) à la philosophie sous-jacente au Projet de Documentation.

Ce document est en cours de rédaction et n'est pas terminé. Les sections inachevées sont indiquées par un astérisque - * - qui précède leur nom.

*Version française de Frédéric Haby <frederic.haby@mail.dotcom.fr >.*

# Table des matières

# Liste des exemples

# Préface

## 1. Invites de l'interpréteur de commandes

La table ci-dessous donne les invites par défaut du système pour un utilisateur normal et pour le super-utilisateur. Elles sont utilisées dans les exemples pour indiquer quel utilisateur doit appliquer l'exemple.

| Utilisateur | Invite |
|---|---|
| Utilisateur normal | `%` |
| `root` | `#` |

## 2. Conventions Typographiques

La table ci-dessous décrit les conventions typographiques utilisées dans le présent ouvrage.

| Signification | Exemples |
|---|---|
| Noms de commandes, fichiers et répertoires. Affichage à l'écran de l'ordinateur. | Modifiez votre fichier `.login`.<br><br>Utilisez `ls -a` pour avoir la liste de tous les fichiers.<br><br>`Vous avez reçu du courrier.` |
| Ce que vous tapez, par opposition à ce que l'ordinateur affiche. | `% su`<br>`Password:` |
| Références aux pages de manuel | Utilisez su(1) pour changer de nom d'utilisateur. |
| Noms d'utilisateurs et de groupes | Seul `root` peut le faire. |
| Mise en valeur | Vous *devez* le faire. |
| Variables sur la ligne de commande ; à remplacer par le nom ou la valeur effectif. | Pour supprimer un fichier, tapez `rm nom_du_fichier` . |
| Variables d'environnement | `$HOME` est votre répertoire utilisateur. |

## 3. Notes, avertissements et exemples

Dans le cours du texte, il peut y avoir des notes, des avertissements et des exemples.

### Note

Les notes apparaissent comme ceci, et contiennent des informations que vous devriez prendre en considération, parce qu'elles peuvent avoir une incidence sur ce que vous faites.

### Avertissement

Les avertissements apparaissent comme ceci, et vous préviennent de problèmes potentiels si vous n'appliquez pas ces instructions. Des dégats peuvent être causés à votre matériel, ou ne pas être physiques, suppression inopinée de fichiers importants par exemple.

### Exemple 1. Un exemple d'exemple

Les exemples apparaissent comme ceci, et sont généralement des exemples que vous devriez tester ou qui vous montrent quels doivent être les résultats d'une opération donnée.

## 4. Remerciements

Mes remerciements à Sue Blake, Patrick Durusau, Jon Hamilton, Peter Flynn et Christopher Maden, qui ont pris le temps de lire les premières versions de ce document et ont apporté de nombreux commentaires et critiques utiles.

# Chapitre 1. Introduction

Bienvenue au Projet de Documentation de FreeBSD. Une documentation de bonne qualité est un facteur important de succès pour FreeBSD, et le Projet de Documentation de FreeBSD - "*The FreeBSD Documentation Project*" - est la source d'une grande part de cette documentation. Votre participation est importante.

L'objectif principal de ce document est d'expliquer clairement *comment est organisé le FDP*, *comment écrire et soumettre de la documentation au FDP* et *comment utiliser les outils disponibles pour produire de la documentation*.

La participation de chacun au FDP est bienvenue. Il n'y a pas de cotisation minimum, pas de quota de documentation à produire par mois. Il vous suffit de vous inscrire sur la liste de diffusion du groupe de documentation de FreeBSD.

Après avoir lu ce document, vous :

- Saurez quelles sont les documentations maintenues par le FDP.
- Serez capable de lire et comprendre le code SGML source des documentations maintenues par le FDP.
- Serez capable de modifier la documentation.
- Saurez comment soumettre vos modifications pour qu'elles soient revues et incorporées à la documentation de FreeBSD.

## 1.1. Le jeu de documentations de FreeBSD

Le FDP a la responsabilité de quatre catégories de documents.

Les pages de manuel
: Les pages de manuel système en anglais ne sont pas rédigées par le FDP, puisqu'elles font partie du système de base. Le FDP, néanmoins, peut (et a) récrit des pages de manuel existantes pour les clarifier ou corriger des inexactitudes.

  Les équipes de traductions sont responsables de la traduction des pages de manuel dans les différentes langues. Ces traductions sont archivées par le FDP.

FAQ
: L'objectif de la FAQ est de répondre (sous forme de courtes questions/réponses) aux questions qui sont posées, ou devraient être posées, sur les différentes listes de diffusion et forums de discussion consacrées à FreeBSD. Son format n'autorise pas de réponses longues et exhaustives.

Manuel de référence - "*Handbook*"
: Le Manuel cherche à être la ressource en ligne exhaustive et de référence pour les utilisateurs de FreeBSD.

Le site Web
: C'est le point de présence central de FreeBSD sur le *World Wide Web*, dont le site principal est http://www.freebsd.org/ et qui a de nombreux sites miroirs dans le monde. Pour beaucoup de gens, ce site est leur première rencontre avec FreeBSD.

Ces quatres catégories de documentation sont disponibles dans les archives CVS de FreeBSD. Ce qui signifie que les modifications et les notifications sont accessibles à tous, et que chacun peut utiliser un programme comme CVSup ou CTM pour maintenir à jour son propre exemplaire local.

En plus de ces documents, de nombreuses personnes ont écrit des guides ou réalisé des sites Web se rapportant à FreeBSD. Certains sont aussi archivés dans l'arborescence CVS (quand l'auteur a donné son accord). Dans d'autre cas, l'auteur a choisi de conserver ses documentations en dehors des archives FreeBSD. Le FDP essaie de donner le maximum de liens possible sur ces documents.

## 1.2. Avant de commencer

Ce document fait l'hypothèse que vous savez déjà :

- Vous procurer et tenir à jour une copie locale de la documentation. Soit en maintenant une copie locale des archives CVS de FreeBSD (avec CVS, CVSup ou CTM), ou en vous servant de CVSup pour ne télécharger que la version extraite - *courante*.
- Comment télécharger et installer de nouveaux logiciels en vous servant soit du catalogue des logiciels de FreeBSD soit de pkg_add(1).

# Chapitre 2. Outils

Le FDP utilise un certain nombre de logiciels pour faciliter la gestion de la documentation de FreeBSD, la convertir en différents formats, et ainsi de suite. Vous devrez vous-même vous servir de ces outils, si vous travaillez à la documentation de FreeBSD.

Tous ces outils existent sous forme de logiciels portés ou pré-compilés pour FreeBSD, ce qui vous simplifie beaucoup la tâche de leur installation.

Vous devrez les installer avant de mettre en pratique les exemples donnés dans les chapitres suivants. Ces chapitres vous expliquent comment vous servir de ces outils.

## Utilisez **`textproc/docproj`** si possible

Vous pouvez gagner beaucoup de temps si vous les installez avec `textproc/docproj`. C'est un *méta-port* qui ne contient pas lui-même de logiciels. Au lieu de cela, il dépend de l'installation correcte de divers autres logiciels portés. Son installation *devrait* télécharger et installer automatiquement tous les paquetages listés dans ce chapitre dont vous aurez besoin, s'ils n'existent pas déjà sur votre système.

L'un des paquetages dont vous aurez peut-être besoin est le jeu de macro-instructions JadeTeX. Celui-ci, à son tour, a besoin que TeX soit installé. TeX est un paquetage volumineux, dont vous n'aurez besoin que si vous voulez générer les versions PostScript et PDF.

Pour économiser du temps et de l'espace disque, vous pouvez préciser si vous voulez ou non installer JadeTeX (et donc TeX). Faites soit :

```
# make JADETEX=yes install
```

ou :

```
# make JADETEX=no install
```

selon le cas.

# 2.1. Outils indispensables

## 2.1.1. Logiciels

Vous aurez besoin de ces programmes avant de pouvoir utilement travailler sur la documentation de FreeBSD. Ils font tous partie de `textproc/docproj`.

SP (`textproc/sp`)
: Une série d'applications, dont un analyseur syntaxique SGML et un outil de normalisation du source SGML.

Jade (`textproc/jade`)
: Une implémentation des DSSSL. Cela sert à convertir des documents marqués vers d'autres formats, dont HTML et TeX.

Tidy (`www/tidy`)
: Un formateur HTML, qui sert à remettre en forme le code HTML généré automatiquement pour qu'il soit plus lisible.

Lynx (`www/lynx-current`)
: Navigateur WWW en mode texte, lynx(1) peut aussi convertir des fichiers HTML en fichiers texte.

## 2.1.2. DTDs et Entités

Ce sont les DTDs et jeux d'entités utilisés par le FDP. Il faut les installer avant de pouvoir travailler à la documentation.

DTD HTML (`textproc/html`)
: HTML est le langage principal du *World Wide Web*, il est utilisé constamment par le site Web de FreeBSD.

DTD LinuxDoc (`textproc/linuxdoc`)
: Une partie de la documentation de FreeBSD est marquée avec LinuxDoc. Le FDP migre activement de LinuxDoc à DocBook.

DTD DocBook (`textproc/docbook`)
: DocBook est conçu pour le marquage de documentation technique et le FDP est en cours de migration de LinuxDoc à DocBook. A la date de rédaction de cette documentation, celle-ci et le Manuel de Référence de FreeBSD sont au format DocBook.

Entités ISO 8879 (`textproc/iso8879`)
: 19 de jeux de caractères ISO 8879:1986 utilisés par de nombreuses DTDs. Cela comprend des symboles mathématiques nommés, les caractères “latins” supplémentaires (accents, signes diacritiques et ainsi de suite) et les lettres grecques.

### 2.1.3. Feuilles de style

Les feuilles de style sont utilisées pour convertir et formater la documentation pour l'affichage à l'écran, l'impression, et ainsi de suite.

Les *Modular DocBook Stylesheets* (`textproc/dsssl-docbook-modular` )
: Les *Modular DocBook Stylesheets* sont utilisées pour convertir la documentation marqué en DocBook aux autres formats, comme HTML ou RTF.

## 2.2. Outils facultatifs

Il n'est pas obligatoire d'installer les outils qui suivent. Si vous le faites cependant, vous trouverez peut-être plus facile de travailler à la documentation et ils vous donneront plus de possibilité de choix du format de sortie.

### 2.2.1. Logiciels

JadeTeX et teTeX (`print/jadetex` et `print/teTeX` )
: Jade et teTeX servent à convertir les documents DocBook en DVI, Postscript et PDF. Il faut pour cela les macro-instructions JadeTeX.

  Si vous n'avez pas l'intention de convertir votre documentation à l'un de ces formats (i.e., HTML, texte et RTF vous suffisent), il n'est pas nécessaire d'installer JadeTeX et teTeX. Cela vous fera gagner du temps et de l'espace disque, teTeX, en effet occupe plus de 30 Mo.

**Important**

Si vous choisissez d'installer JadeTeX et teTeX, vous devrez configurer teTeX après avoir installé JadeTeX. `print/jadetex/pkg-message` vous donnera des instructions détaillées sur la façon de procéder.

Emacs ou xemacs (`editors/emacs` ou `editors/xemacs`)
: Ces deux éditeurs ont un mode spécialisé pour travailler sur des documents marqués conformément à une DTD SGML. Cela vous permet d'avoir moins de choses à saisir et limite la possibilité d'erreurs.

  Vous n'êtes pas obligé de les utiliser, n'importe quel éditeur peut servir avec des documents marqués. Mais vous trouverez peut-être qu'ils vous permettent d'être plus efficace.

Si quelqu'un a d'autres outils utiles pour manipuler des documents SGML, merci de transmettre l'information à Nik Clayton (nik@freebsd.org), de façon à ce qu'il les ajoute à cette liste.

# Chapitre 3. Introduction à SGML

La majorité des documentations du FDP utilisent SGML. Ce chapitre vous explique ce que cela signifie exactement, comment lire et comprendre le source de la documentation et décrit la façon d'utiliser le SGML que vous recontrerez dans la documentation.

Des parties de cette section se sont inspirées du livre de Mark Galassi, *"Get Going With DocBook"*.

## 3.1. Introduction

Il était autrefois facile de travailler sur des documents électroniques. Vous n'aviez normalement à connaître que le jeu de caractères utilisé (ASCII, EBCDIC, ou l'un des nombreux autres) et c'était à peu près tout. Le texte était du texte, et vous voyiez vraiment ce que vous obteniez. Pas de sophistication, pas de formatage, pas d'intelligence.

Cela devint inévitablement insuffisant. Une fois que vous avez du texte qu'une machine peut lire, vous vous attendez à ce que la machine puisse l'utiliser et le manipuler intelligemment. Vous aimeriez pouvoir préciser que certaines phrases sont accentuées, y ajouter un glossaire ou des hyper-liens. Vous voulez que les noms de fichiers apparaissent en police "machine à écrire" à l'écran et en italique à l'impression, et tout un tas d'autres options de présentation encore.

Il fut un temps où l'on pensait que l'Intelligence Artificielle (IA) rendrait cela facile. Votre ordinateur pourrait lire le document et identifier les phrases clés, les noms de fichiers, le texte que l'utilisateur devait taper, et d'autres encore. Malheureusement, la réalité est un peu différente, et il faut aider nos ordinateurs à manipuler intelligemment notre texte.

Plus précisement, il faut les aider à indentifier ce qui est quoi. Vous et moi, à la vue de :

> Pour effacer `/tmp/foo` , utilisez rm(1) :
>
> ```
> % rm /tmp/foo
> ```

distinguons facilement ce qui est nom de fichier, commande à taper, référence aux pages de manuel, et ainsi de suite. Mais l'ordinateur lui ne le peut pas. Pour cela, Nous avons besoin des marques.

Le "marquage" est communément qualifié de "valeur ajoutée" ou "coût augmenté". Le terme prend ces deux sens quand il s'applique au texte. La marquage est du texte en supplément dans le document, distinct par un moyen ou un autre du contenu du document,

de façon à ce que les programmes qui traitent le document puisse le lire et l'utiliser pour prendre des décisions. Les éditeurs peuvent masquer le marquage à l'utilisateur, de façon à ce qu'il ne soit pas perturbé par ces marques.

L'information supplémentaire donnée avec les marques *ajoute de la valeur* au document. Le marquage doit habituellement être manuel - après tout, si les ordinateurs pouvait analyser suffisamment le texte pour ajouter les marques, il n'y en aurait alors en fait pas besoin. Cela *augment le coût* du document.

L'exemple précédent est codé comme suit dans le présent document :

```
<para>Pour effacer <filename>/tmp/foo</filename>, utilisez
  &man.rm.1;.</para>

<para><command>rm /tmp/foo</command></para>
```

Comme vous pouvez le constater, le marquage est clairement séparé du contenu.

Bien évidemment, si vous devez utiliser des marques, vous devrez définir ce que les marques veulent dire et comment elles doivent être traitées. Il vous faudra un language de marquage auquel vous référer pour marquer vos documents.

Un seul language de marquage peut bien sûr ne pas suffire. Les besoins de marquage d'une documentation technique diffèrent énormément de ceux de recettes de cuisines. ces derniers seront à leur tour différents de ceux d'un language de marquage pour de la poésie. Vous avez en fait besoin d'un language qui vous permette de définir ces autres languages de marquage. Un *méta-language de marquage*.

C'est exactement ce qu'est *Standard Generalised Markup Language (SGML)* - Language de Marquage Standard Généralisé. De nombreux languages de marquage sont écrits en SGML, dont les deux languages les plus utilisés par le FDP, HTML et DocBook.

Chaque définition d'un language s'appelle plus exactement une *Document Type Definition (DTD)* - Définition de Type de Document. La DTD définit les noms des éléments utilisables, leur ordre d'apparition (et leur hiérarchie) et les informations qui s'y rapportent. Une DTD est parfois désignée comme une *application* de SGML.

Une DTD est une spécification *complète* de tous les éléments autorisés, de l'ordre dans lequel ils doivent être utilisés, quels sont ceux qui sont obligatoires, quels sont ceux qui sont facultatifs, et ainsi de suite. Il est alors possible d'écrire un *analyseur* qui lise et la DTD et le document qui prétend s'y conformer. L'analyseur peut alors vérifier si tous les éléments requis sont bien présents dans l'ordre voulu dans le document et s'il y a des erreurs dans le marquage. On appelle habituellement cela « valider le document ».

### Note

Ce traitement ne valide uniquement que le choix des éléments, leur ordre, et ainsi de suite, se conforme à ce que définit la DTD. Il ne vérifie *pas* que vous avez utilisé les marques *appropriées* au document. Si vous marquez tous les noms de fichiers de votre document comme des noms de fonctions, l'analyseur ne le signalera pas comme une erreur (en supposant, bien sûr, que votre DTD définisse des éléments pour les noms de fichiers et de fonctions et qu'ils aient le droit d'apparaître aux mêmes endroits).

Il est probable que vos contributions au Projet de Documentation consiste en documents marqués soit en HTML soit en DocBook, plutôt qu'en modifications aux DTDs. Pour cette raison, cet ouvrage n'abordera pas la façon d'écrire une DTD.

## 3.2. Eléments, marques et attributs

Toutes les DTDs écrites en HTML ont des caractéristiques communes. Ce n'est guère surprenant comme le montre inévitablement la philosophie qui sous-tend SGML. Une des manifestations les plus visibles de cette philosophie est la caractérisation en *contenu* et *éléments*.

Votre documentation (que ce soit une seule page Web ou un ouvrage volumineux) est vue comme étant un contenu. Ce contenu est alors divisé (et ensuite subdivisé) en éléments. L'objectif de l'ajout de marques est de nommer et de définir le début et la fin de ces éléments pour traitement ultérieur.

Considérez par exemple un livre type. Au plus haut niveau, ce livre lui-même est un élément. Cet élément “livre” contient évidemment des chapitres, qui peuvent aussi être légitimement considérés comme des éléments. Chaque chapitre contiendra à son tour des éléments, tels que des paragraphes, des citations et de notes de bas de page. Chaque paragraphe peut lui-même contenir encore des éléments, pour identifier le texte parlé par exemple, ou les noms des personnages de l'histoire.

Vous pouvez si vous le voulez voir cela comme un “morcelement” du contenu. A la racine, vous avez un morceau, le livre. Un niveau en dessous, vous avez plus de morceaux, les chapitres individuels. Ils sont à leur tour morcelés en pargraphes, notes de bas de page, noms des personnages, et ainsi de suite.

Remarquez que vous pouvez différencier les éléments sans utiliser la terminologie SGML. C'est vraiment immédiat. Vous pouvez le faire avec un surligneur et un livre imprimé, en utilisant des couleurs différentes pour chaque type d'élément.

Nous n'avons bien sûr pas de surligneur électronique, il nous faut donc un autre moyen d'indiquer à quel élément appartient chaque morceau du contenu. Dans les languages écrits avec SGML ,(HTML, DocBook, et al.), cela se fait avec des *marques*.

Une marque sert à dire où commence et où finit un élément. *La marque ne fait pas partie de l'élément lui-même*. Comme chaque DTD est habituellement écrite pour marquer des types d'informations spécifiques, chacune reconnaîtra des éléments différents, et aura donc des noms différents pour les marques.

Pour un élément appelé *`nom-de-l'élément`* , la marque de début sera normalement `<nom-de-l'élément>` . La marque de fin correspondante sera `</nom-de-l'élément>` .

### Exemple 3.1. Utiliser un élément (marques de début et de fin)

HTML dispose d'un élément pour indiquer que le contenu inclus est un paragraphe, appelé `p`. Cet élément a une marque de début et une de fin.

```
<p>C'est un paragraphe. Il commence avec la marque de début ↺
pour
  l'élément 'p', et se terminera avec la marque de fin pour
  l'élément 'p'</p>

<p>C'est un autre paragraphe. Mais il est beaucoup plus
  court.</p>
```

Tous les éléments n'ont pas besoin d'une marque de fin. Certains n'ont pas de contenu. En HTML, par exemple, vous pouvez indiquer que vous voulez avoir une ligne horizontal dans votre document. Cette ligne n'a bien sûr aucun contenu, vous n'avez donc besoin que de la marque de début pour cet élément.

### Exemple 3.2. Utiliser un élément (marque de début uniquement)

HTML dispose d'un élément pour inclure une ligne horizontale, appelé `hr`. C'est un élément sans contenu, il n'a donc qu'une marque de début.

```
<p>C'est un paragraphe.</p>

<hr>

<p>C'est un autre paragraphe. Une ligne horizontale le sépare
  du précédent.</p>
```

Si ce n'était pas encore clair, les éléments peuvent contenir d'autres éléments. Dans l'exemple du livre plus haut, ce livre contenait tous les chapitres, qui à leur tour contenaient tous les paragraphes, et ainsi de suite.

### Exemple 3.3. Eléments dans des éléments ; `em`

```
<p>C'est un <em>paragraphe</em> simple où certains
  <em>mots</em> ont été <em>mis en valeur</em>.</p>
```

La DTD définira les règles qui disent quels éléments peuvent être inclus dans quels autres éléments, et ce qu'ils peuvent précisement contenir.

### Important

Les gens confondent souvent marques et éléments comme si c'étaient des termes interchangeables. Ce n'est pas le cas.

Un élément est une partie de la structure d'un document. Un élément a un début et une fin. Les marques définissent où commence et où finit le document.

Quand le présent document (ou quelqu'un d'autre qui connait le SGML) parle de la marque “the <p> tag”, cela se rapporte au texte composé des trois caractères <, p et >. Mais la phrase “l'élément <p>” désigne tout l'élément.

Cette distinction *est* très subtile. Mais gardez la à l'esprit.

Les éléments peuvent avoir des attributs. Un attribut a un nom et une valeur, et sert à donner des informations supplémentaires concernant l'élément. Ce peuvent être des in-

formations qui précisent comment l'élément doit être formaté, ou un identifiant unique pour cette occurrence de l'élément, ou autre chose encore.

Les attributs d'un élément sont donnés *dans* la marque de début de l'élément et ont la forme `nom-de-l'attribut="valeur-de-l'attribut"` .

Dans les versions récentes d'HTML, l'élément `p` a un attribut appelé `align`, qui suggère un alignement (justification) du paragraphe au programme affichant l'HTML.

L'attribut `align` peut prendre l'une des quatre valeurs prédéfinies, `left`, `center`, `right` et `justify`. Si l'attribut n'est pas précise, la valeur par défaut est `left`.

### Exemple 3.4. Utiliser un élément avec un attribut

```
<p align="left">L'attribut align est superflus pour ce ↺
paragraphe,
  puisque 'left' est la valeur par défaut.</p>

<p align="center">Ce paragraphe sera peut-être centré.</p>
```

Certains attributs ne prennent que des valeurs prédéfinies, comme `left` ou `justify`. D'autres peuvent prendre les valeurs que vous voulez. Si vous avez besoin de quotes (") dans un attribut, mettez la valeur de l'attribut entre simples quotes.

### Exemple 3.5. Simples quotes dans un attribut

```
<p align='right'>Je suis &agrave; droite !</p>
```

Vous n'avez pas toujours besoin de mettre la valeur de l'attribut entre simples quotes. Les régles à ce sujet sont cependant subtiles, et il est beaucoup plus simple de *toujours* mettre entre simples quotes les valeurs des attributs.

## 3.2.1. A faire...

Pour tester les exemples donnés dans ce document, vous devrez installer des logiciels sur votre système et vérifiez qu'une variable d'environnement est correctement définie.

1. Téléchargez et installez `textproc/docproj` du catalogue des logiciels portés de FreeBSD. C'est un *méta-port* qui doit télécharger et installer tous les programmes et fichiers utilisés par le Projet de Documentation.

2. Ajoutez les lignes pour définir `SGML_CATALOG_FILES` à vos procédures d'initialisation de l'interpréteur de commandes.

Exemple 3.6. **`.profile`**, pour les utilisateurs de sh(1) et bash(1)

```
SGML_ROOT=/usr/local/share/xml
SGML_CATALOG_FILES=${SGML_ROOT}/jade/catalog
SGML_CATALOG_FILES=${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/docbook/catalog:
$SGML_CATALOG_FILES
export SGML_CATALOG_FILES
```

Exemple 3.7. **`.login`**, pour les utilisateurs de csh(1) et tcsh(1)

```
setenv SGML_ROOT /usr/local/share/xml
setenv SGML_CATALOG_FILES ${SGML_ROOT}/jade/catalog
setenv SGML_CATALOG_FILES ${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/docbook/catalog:
$SGML_CATALOG_FILES
```

Déconnectez-vous et reconnectez-vous ensuite, ou exécutez ces commandes pour définir la variable d'environnement.

3. Créez un fichier `exemple.xml` , où vous mettrez :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0
```

```
  Transitional//EN">

<html>
  <head>
    <title>Exemple de fichier HTML</title>
  </head>

  <body>
    <p>C'est un paragraphe avec du texte.</p>

    <p>C'est encore un paragraphe avec du texte.</p>


    <p align="right">Ce paragraphe sera peut-être justifié ↺
&agrave;
      droite.</p>
  </body>
</html>
```

4. Essayez de le valider avec un analyseur syntaxique SGML.

   L'analyseur syntaxique [8] nsgmls(1) fait partie de `textproc/docproj` . nsgmls(1) lit normalement un document marqué en utilisant une DTD SGML et génère l'*Element Structure Information Set (ESIS)* - Informations sur la Structuration en Eléments - mais cela ne nous concerne pas pour le moment.

   Néanmoins, avec le paramètre `-s`, nsgmls(1) ne génère rien mais affiche simplement les messages d'erreurs éventuels. C'est utile pour vérifier si votre document est correct ou non.

   Utilisez nsgmls(1) pour vérifier si votre document est valide :

```
% nsgmls -s example.xml
```

   Vous constaterez que nsgmls(1) n'affiche rien. Cela signifie qu'il a validé votre document.

5. Voyez ce qui ce passe si vous oubliez un élément requis. Supprimez les marques `title` et `/title` et relancer la validation.

```
% nsgmls -s example.xml
nsgmls:example.xml:5:4:E: character data is not allowed here
nsgmls:example.xml:6:8:E: end tag for "HEAD" which is not finished
```

   Les messages d'erreur de nsgmls(1) sont structurés en colonnes séparés par des deux-points ou des points-virgules.

| Colonne | Signification |
|---|---|
| 1 | Nom du programme qui a généré l'erreur. Ce sera toujours `nsgmls`. |
| 2 | Nom du fichier où se trouve l'erreur. |
| 3 | Numéro de la ligne où se trouve l'erreur. |
| 4 | Numéro de la colonne où se trouve l'erreur. |
| 5 | Une lettre donnant le type de message d'erreur. `I` pour un message d'information, `W` pour un message d'avertissement, `E` pour un message d'erreur et `X` pour les références croisées. (Ce n'est cependant pas toujours la cinquième colonne. `nsgmls -sv` affiche `nsgmls:I: SP version "1.3"` - selon la version installée. Comme vous pouvez le constater, c'est un message d'information.) Vous voyez donc que nous avons dans notre exemple des messages d'erreurs. |
| 6 | Le texte du message d'erreur. |

Vous aurez plus d'informations sur les erreurs de nsgmls(1) dans la Unofficial 'Kindler, Gentler HTML Validator' FAQ.

Ne pas mettre les marques `title` a généré 2 erreurs différentes.

La première erreur indique que l'analyseur SGML a rencontré un contenu (ici, des caractères, au lieu d'une marque de début d'élément) alors qu'il attendait autre chose. Dans le cas présent, l'analyseur attendait une marque de début pour un élément valide à l'intérieur de `head` (`title` par exemple).

La deuxième erreur est due au fait que les éléments `head` doivent contenir un élément `title`. nsgmls(1) considère alors que l'élément n'est pas complet. La marque de fin indique donc que l'élément se termine alors qu'il n'est pas correctement renseigné.

6. Remettez l'élément `title` en place.

## 3.3. La déclaration DOCTYPE

Au début de chaque document que vous rédigez, vous devez préciser le nom de la DTD à laquelle le document se conforme. Cela pour que les analyseurs syntaxiques SGML la connaissent et puissent valider le document.

Cette information est habituellement donnée sur une seule ligne, dans la déclaration DOCTYPE.

Voici une déclaration typique pour un document conforme à la version 4.0 de la DTD HTML :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN">
```

Cette ligne a plusieurs composants distincts :

`<!`
: C'est l'*indicateur* qui dit que c'est une déclaration SGML. Cette ligne définit le type de document.

`DOCTYPE`
: Précise que c'est la déclaration SGML du type de document.

`html`
: Définit le premier élément qui apparaîtra dans le document.

`PUBLIC "-//W3C//DTD HTML 4.0//EN"`
: Donne le *Formal Public Identifier (FPI)* - Identifiant Public Officiel - de la DTD à laquelle le document se conforme.

  `PUBLIC` n'appartient pas au FPI, mais indique au processeur SGML comment trouver la DTD référencée par le FPI. Les autres façons de dire à l'analyseur SGML comment trouver la DTD sont données plus loin.

`>`
: Retour au document.

### 3.3.1. Formal Public Identifiers (FPIs) - Identifiants Publics Officiels

**Note**

Vous n'avez pas besoin de connaître ce qui suit, mais ce n'est n'est pas inutile, et cela peut vous aider à résoudre des problèmes, si votre processeur SGML ne trouve pas la DTD que vous utilisez.

Les FPIs doivent respecter une syntaxe précise. La voici :

```
"Propriétaire //Mot-Clé  Description //Langue"
```

*Propriétaire*
: Indique qui détient le FPI.

  Si la chaîne de caractères commence par “ISO”, c'est un FPI ISO. Par exemple, le FPI `"ISO 8879:1986//ENTITIES Greek Symbols//EN"` donne `ISO 8879:1986` comme propriétaire du jeu d'entités pour les lettres grecques. ISO 8879:1986 est le numéro ISO du standard SGML.

  Sinon, cette chaîne sera de la forme `-//Propriétaire` ou `+//Propriétaire` (remarquez que la seule différence est le + ou - du début).

  Si la chaîne commence par un -, c'est que le propriétaire n'est pas enregistré, il l'est si elle commence par un +.

  L'ISO 9070:1991 définit comment sont générés les noms enregistrés ; ils peuvent dériver du numéro d'une publication ISO, d'un code ISBN ou d'un code d'organisation affecté selon l'ISO 6523. De plus, il pourrait y avoir une autorité d'enregistrement pour l'affectation de ces noms. Le conseil ISO a délégué cela à l'*American National Standards Institute (ANSI)* - Institut National Américain des Standards.

  Comme le Projet FreeBSD n'est pas enregistré, la chaîne utilisée est `-//FreeBSD`. Comme vous pouvez vous en rendre compte, le W3C n'est pas non plus un propriétaire enregistré.

*Mot-Clé*
: Il y a plusieurs mots-clés qui définissent le type d'information dans le fichier. Les mots-clés les plus courants sont : `DTD`, `ELEMENT`, `ENTITIES` et `TEXT`. `DTD` ne sert que pour les DTD, `ELEMENT` sert habituellement pour les extraits de DTD qui ne contiennent que des entités ou des déclarations d'éléments. `TEXT` sert pour le contenu SGML (texte et marques).

*Description*
: La description que vous souhaitez donner du contenu du fichier. Cela peut inclure des numéros de version et n'importe quel texte court qui ait un sens et soit unique au système SGML.

*Langue*
: C'est une code ISO de deux caractères qui identifie la langue utilisée dans le fichier. Pour l'anglais, c'est `EN`.

#### 3.3.1.1. Fichiers `catalog`

Si vous avez utilisé la syntaxe décrite plus haut et essayé d'utiliser un processeur SGML pour traiter votre document, il aura besoin de convertir le FPI en un nom de fichier sur votre ordinateur qui décrive la DTD.

Vous pouvez pour cela vous servir d'un fichier catalogue (habituellement appelé `catalog`). Il contient des lignes qui donnent les correspondances entre FPIs et noms de fichiers. Par exemple, s'il y a la ligne :

```
PUBLIC "-//W3C//DTD HTML 4.0//EN" "4.0/strict.dtd"
```

le processeur SGML cherchera la DTD dans le fichier `strict.dtd` du sous-répertoire `4.0` où se trouve le fichier `catalog` qui comporte cette ligne.

Jettez un oeil au fichier `/usr/local/share/xml/html/catalog` . C'est le fichier catalogue pour les DTDs HTML qui ont été installées par le logiciel porté `textproc/docproj`.

#### 3.3.1.2. `SGML_CATALOG_FILES`

Pour trouver un fichier `catalog`, votre processeur SGML doit savoir où chercher. La plupart d'entre eux ont des paramètres de leur ligne de commande pour donner le chemin d'accès à un ou plusieurs catalogues.

Vous pouvez par ailleurs définir `SGML_CATALOG_FILES` pour désigner ces fichiers. Cette variable d'environnement doit contenir une liste de fichiers catalogues (donnés par leurs chemins d'accès complets) séparés par des points-virgules.

Habituellement, vous incluerez les fichiers suivants :

- `/usr/local/share/xml/docbook/catalog`
- `/usr/local/share/xml/html/catalog`
- `/usr/local/share/xml/iso8879/catalog`
- `/usr/local/share/xml/jade/catalog`

### 3.3.2. Alternatives aux FPIs

Au lieu d'utiliser un FPI pour préciser la DTD utilisée (et donc le fichier qui contient la DTD), il est possible de donner explicitement le nom du fichier.

La syntaxe pour le faire est légèrement différente :

```
<!DOCTYPE html SYSTEM "/path/to/file.dtd">
```

Le mot-clé `SYSTEM` indique que le processeur SGML doit localiser le fichier d'une façon qui dépend du système. Cela signifie habituellement (mais pas toujours) que la DTD sera définie par un nom de fichier.

Il est préférable d'utiliser des FPIs pour des raisons de portabilité. Vous ne voulez pas livrer un exemplaire de la DTD avec votre document, et si vous avez utilisé l'identifiant `SYSTEM`, il faudra que chacun ait ses DTDs aux mêmes endroits.

## 3.4. Revenir au SGML

On a dit plus haut dans cette introduction que le SGML n'était utilisé que pour écrire les DTDs. Ce n'est pas tout à fait vrai. Il y a des éléments de la syntaxe SGML que vous voudrez pouvoir utiliser dans vos documents. Par exemple, vous pouvez y inclure des commentaires, qui seront ignorés par les analyseurs. Les commentaires sont inclus en utilisant une syntaxe SGML. D'autres utilisations du SGML dans les documents seront mentionnées plus loin.

Il vous faut évidemment un moyen d'indiquer au processeur SGML que ce qui va suivre n'est pas constitué d'éléments du document, mais est du SGML que le processeur doit prendre en compte.

Ces sections sont marqués avec `<! ... >` dans votre document. Tout ce qui se trouve entre ces délimiteurs est du code SGML comme on en trouve dans les DTDs.

Comme vous venez peut-être de vous en rendre compte, la déclaration DOCTYPE est un exemple de syntaxe SGML que vous devez inclure dans votre document...

## 3.5. Commentaires

*** A Traduire ***

Les commentaires suivent une syntaxe SGML et ne sont normalement autorisés que dans une DTD. Cependant comme la Section 3.4, « Revenir au SGML » le montre, il est possible d'inclure du SGML dans vos documents.

Les délimiteurs pour les commentaires SGML sont constitués de la chaîne de caractères "`--`". Une première occurence ouvre le commentaire, et la seconde le ferme.

## Exemple 3.8. Commentaire SGML générique

```
<!-- C'est le texte du commentaire -->

<!-- C'est un autre commentaire -->

<!-- Voici une façon de mettre un commentaire
     sur plusieurs lignes -->

<!-- Voici une autre façon --
  -- de le faire -->
```

Si vous avez déjà utilisé HTML auparavant, on vous a peut-être donné des règles différentes pour les commentaires. En particulier, vous pensez peut-être qu'ils commencent par `<!--` et ne se terminent qu'avec `-->`.

Ce n'est *pas* le cas. Les analyseurs syntaxiques de nombreux navigateurs sont défectueux et acceptent cette syntaxe. Ceux qu'utilisent le Projet de Documentation sont plus rigoureux et rejetteront les documents qui comportent cette erreur.

## Exemple 3.9. Commentaires SGML erronnés

```
>!-- C'est en commentaire --

     CE N'EST PAS EN COMMENTAIRE!

  -- retour au commentaire -->
```

L'analyseur SGML traitera cela comme s'il trouvait :

```
<!CE N'EST PAS EN COMMENTAIRE>
```

Ce qui n'est pas du SGML valide et donnera des messages d'erreur source de confusion.

```
<!----- C'est un très mauvaise idée ----->
```

Comme l'exemple le suggère, ne mettez *pas* de commentaires de ce type.

```
<!--==================================================-->
```

C'est une (légèrement) meilleure idée, mais c'est toute de même une source de confusion potentielle pour les débutants en SGML.

### 3.5.1. A faire...

1. Ajoutez des commentaires à `exemple.xml` et validez vos modifications avec nsgmls(1)
2. Ajoutez des commentaires incorrects à `exemple.xml` , pour voir quels messages d'erreur produit alors nsgmls(1).

## 3.6. Entités

Les entités fournissent un mécanisme pour désigner des parties d'un contenu. Lorsque l'analyseur SGML traite votre document, il remplace les entités qu'il rencontre par le contenu de ces entités.

C'est un bon moyen pour avoir du texte réutilisable et facile à modifier. C'est aussi le seul moyen d'inclure, en utilisant SGML, un fichier marqué dans un autre.

Il y a deux sortes d'entités SGML qui s'utilisent dans des situations différentes : les *entités générales* et les *entités paramètres*.

### 3.6.1. Entités Générales

Vous ne pouvez pas employer les entités générales dans un contexte SGML (bien que ce soit là que vous les définissiez). Elles ne peuvent être utilisées que dans votre document. Comparez cela au cas des entités paramètres.

Chaque entité générale a un nom. Quand vous voulez y faire référence (et donc inclure le texte qu'elle contient dans votre document), vous mettez `&nom-de-l'entité;` . Supposons par exemple que vous ayez une entité appelée `version.courante` qui contienne le numéro de version courante de votre produit. Vous pourriez écrire :

```
<para>La version courante de notre produit est la
  &version.courante;.</para>
```

Quand le numéro de version change, il vous suffit de modifier la définition de l'entité générale et de recompiler votre document.

Vous pouvez aussi vous servir d'entités générales pour représenter des caractères que vous ne pouvez pas inclure autrement dans un document SGML. < et &, par exemple, ne doivent normalement pas apparaître dans un document SGML. Quand l'analyseur SGML rencontre un symbole <, il suppose qu'il précède une marque (de début ou de fin), et quand il rencontre un symbole &, il suppose que le texte qui le suit est le nom d'une entité.

Heureusement, il y a deux entités générales, < et & pour le cas où vous auriez besoin d'inclure l'un ou l'autre de ces symboles.

Une entité générale ne peut être définie que dans un contexte SGML. On le fait habituellement immédiatement après la déclaration DOCTYPE.

### Exemple 3.10. Définition d'entités générales

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY version.courante    "3.0-RELEASE">
<!ENTITY derniere.version  "2.2.7-RELEASE">
]>
```

Remarquez que la déclaration DOCTYPE est suivie d'un crochet ouvrant à la fin de la première ligne. Les deux entités sont définies aux deux lignes suivantes, avant le crochet fermant. La déclaration DOCTYPE se termine ensuite.

Les crochets sont nécessaires pour dire que nous ajoutons un complément à la DTD mentionnée par la déclaration DOCTYPE.

## 3.6.2. Entités paramètres

Comme les entités générales, les entités paramètres servent à nommer des parties réutilisables du texte. Cependant, alors que les entités générales peuvent être utilisées dans le corps du document, les entités paramètres ne peuvent être employées que dans un contexte SGML.

Les entités paramètres sont définies de la même manière que les entités générales. Sinon qu'au lieu de vous servir de `&inomd-de-l'entité;` pour y faire référence, vous utiliserez `%nom-de-l'entité;` [1]. Leur définition comporte aussi un `%` entre le mot-clé `ENTITY` et le nom de l'entité.

[1] Les entités *P*aramètres employent le symbole *P*ourcent.

Exemple 3.11. Définition d'entités paramètres

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % param.du "du">
<!ENTITY % param.texte "text">
<!ENTITY % param.encore  "encore %param.du more %param.texte">
]>
```

Cela ne paraît peut être pas très utile. On verra pourtant que ça l'est.

### 3.6.3. A faire...

1. Définissez un entité générale dans `exemple.xml` .

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" [
<!ENTITY version "1.1">
]>

<html>
  <head>
    <title>Exemple de fichier HTML</title>
  </head>

  <body>
    <p>C'est un paragraphe avec du texte.</p>

    <p>C'est encore un paragraphe avec du texte.</p>

    <p align="right">Ce paragraphe sera peut-être justifié ↺
&agrave;
      droite</p>

    <p>La version courante de ce document est : &version;</p>
  </body>
</html>
```

2. Validez le document avec nsgmls(1)

3. Chargez `exemple.xml` avec votre navigateur (vous devrez peut-être le recopier dans `exemple.html` pour que votre navigateur le reconnaisse comme un document HTML).

A moins que votre navigateur ne soit très évolué, il ne remplacera pas la référence `&version;` à l'entité par le numéro de version. Les analyseurs de la plupart des navigateurs sont élémentaires et ne gèrent pas correctement le SGML[2].

4. La solution est de *normaliser* votre document avec un outil de normalisation du SGML. Ce type d'outil lit un document SGML valide et le transforme en un autre document SGML tout aussi valide. En particulier, il y remplace les références aux entités par leur contenu.

   Vous pouvez le faire avec sgmlnorm(1).

   ```
   % sgmlnorm exemple.xml > exemple.html
   ```

   `exemple.html` doit maintenant contenir une version normalisée (i.e., où les références aux entités ont été remplacées par leur contenu) de votre document, prête à être affichée par votre navigateur.

5. Si vous jetez un oeil au résultat de sgmlnorm(1), vous verrez qu'il ne comporte pas de déclaration DOCTYPE au début. Pour qu'elle y soit, utilisez l'option `-d` :

   ```
   % sgmlnorm -d exemple.xml > exemple.html
   ```

## 3.7. Utiliser les entités pour inclure des fichiers

Les entités (générales et paramètres) sont particulièrement utiles pour inclure un fichier dans un autre.

### 3.7.1. Utiliser les entités générales pour inclure des fichiers

Supposons que le contenu d'un livre SGML soit découpé en fichiers, à raison d'un fichier par chapitre, appelés `chaptitre1.xml` , `chapitre2.xml` , et ainsi de suite, et que le fichier `livre.xml` inclue ces chapitres.

Pour que vos entités aient pour valeur le contenu de ces fichiers, vous les déclarerez avec le mot-clé `SYSTEM`. Cela indique à l'analyseur SGML qu'il doit utiliser le contenu du fichier mentionné comme valeur de l'entité.

Exemple 3.12. Utiliser les entités générales pour inclure des fichiers

[2] C'est tout à fait dommage. Imaginez les problèmes et bricolages (comme les *Server Side Includes*) que cela éviterait.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY chapitre.1 SYSTEM "chapitre1.xml">
<!ENTITY chapitre.2 SYSTEM "chapitre2.xml">
<!ENTITY chapitre.3 SYSTEM "chapitre3.xml">
]>

<html>
  &chapitre.1;
  &chapitre.2;
  &chapitre.3;
</html>
```

### Avertissement

Quand vous vous servez d'entités générales pour inclure d'autres fichiers dans un document, les fichiers inclus (`chapitre1.xml` , `chapitre2.xml` , et ainsi de suite) ne doivent *pas* commencer par une déclaration DOCTYPE. Ce serait une erreur de syntaxe.

## 3.7.2. Utiliser les entités paramètres pour inclure des fichiers

Rappelez-vous que les entités paramètres ne peuvent être utilisées que dans un contexte SGML. Quand aurez-vous besoin d'inclure un fichier dans un contexte SGML ?

Vous pouvez vous en servir pour être sûr de pouvoir réutiliser vos entités générales.

Supposons que votre document comporte de nombreux chapitres, et que vous réutilisiez ces chapitres dans deux livres différents, chacun organisant ces chapitres de façon différente.

Vous pourriez donner la liste des entités en tête de chaque livre, mais cela pourrait rapidement devenit fastidieux à gérer.

Mettez, au lieu de cela, les définitions des entités générales dans un fichier, et utilisez une entité paramètre pour inclure ce fichier dans votre document.

### Exemple 3.13. Utiliser les entités paramètres pour inclure des fichiers

Mettez d'abord les définitions de vos entités dans un fichier séparé, appelé `chapitres.ent` . Voici ce qu'il contiendra :

```
<!ENTITY chapitre.1 SYSTEM "chapitre1.xml">
<!ENTITY chapitre.2 SYSTEM "chapitre2.xml">
<!ENTITY chapitre.3 SYSTEM "chapitre3.xml">
```

Créez maintenant une entité paramètre qui fasse référence au contenu de ce fichier. Utilisez ensuite cette entité pour inclure le fichier dans votre document, vous pourrez alors y utiliser les entités générales. Ce que vous faites de la même façon que précédemment :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % chapitres SYSTEM "chapitres.ent">
%chapitres;
]>

<html>
  &chapitre.1;
  &chapitre.2;
  &chapitre.3;
</html>
```

## 3.7.3. A faire...

### 3.7.3.1. Utiliser les entités générales pour inclure des fichiers

1. Créez trois fichiers, `para1.xml` , `para2.xml` et `para3.xml` .

   Mettez-y quelque chose qui ressemble à ceci :

   ```
   <p>C'est le premier paragraphe.</p>
   ```

2. Modifiez `exemple.xml` de la façon suivante :

   ```
   <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
   <!ENTITY version "1.1">
   <!ENTITY para1 SYSTEM "para1.xml">
   <!ENTITY para2 SYSTEM "para2.xml">
   <!ENTITY para3 SYSTEM "para3.xml">
   ]>

   <html>
     <head>
       <title>Exemple de fichier HTML</title>
     </head>

     <body>
   ```

```
    <p>La version courante de ce document est : &version;</p>

    &para1;
    &para2;
    &para3;
  </body>
</html>
```

3. Générez `exemple.html` en normalisant `exemple.xml` .

```
% sgmlnorm -d exemple.xml > exemple.html
```

4. Affichez `exemple.html` avec votre navigateur Web et vérifiez que les fichiers `paran.xml` ont bien été inclus dans `exemple.html` .

### 3.7.3.2. Utiliser les entités paramètres pour inclure des fichiers

Note

Vous devez d'abord avoir mis en pratique l'exemple précédent.

1. Modifiez comme ceci `exemple.xml` :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % entites SYSTEM "entites.xml"> %entites;
]>

<html>
  <head>
    <title>Exemple de fichier HTML</title>
  </head>

  <body>
    <p>La version courant de ce document est : &version;</p>

    &para1;
    &para2;
    &para3;
  </body>
</html>
```

2. Créez un nouveau fichier, `entites.xml` , qui contienne :

```
<!ENTITY version "1.1">
<!ENTITY para1 SYSTEM "para1.xml">
<!ENTITY para2 SYSTEM "para2.xml">
```

```
<!ENTITY para3 SYSTEM "para3.xml">
```

3. Générez `exemple.html` en normalisant `exemple.xml` .

```
% sgmlnorm -d exemple.xml > exemple.html
```

4. Affichez `exemple.html` avec votre navigateur Web et vérifiez que les fichiers `paran.xml` ont bien été inclus dans `example.html` .

# 3.8. Sections marquées

SGML fournit un mécanisme pour définir quelles parties d'un document doivent être traitées de façon particulière. On appelle cela des “sections marquées”.

### Exemple 3.14. Structure d'une section marquée

```
<![ MOT-CLE [
  Contenu de la section marquée
]]>
```

Comme vous pouviez vous y attendre, une section marquée est une fonctionnalité SGML et commence donc par `<!`.

Le premier crochet ouvrant délimite la section marquée.

Le *`MOT-CLE`* définit comment cette section marquée doit être traitée par l'analyseur.

Le second crochet ouvrant indique que le contenu de la section marquée commence là.

La section marquée se termine par deux crochets fermants, puis un `>` pour indiquer que l'on quitte le contexte SGML et que l'on revient au document.

## 3.8.1. Mots-clés pour les sections marquées

#### 3.8.1.1. `CDATA`, `RCDATA`

Ces deux mots-clés définissent des sections marquées comme *modèle de contenu* et vous permettent de modifier sa valeur par défaut.

Quand un analyseur SGML traite un docuemnt, il mémorise ce que l'on appelle le “modèle de contenu”.

En bref, le modèle de contenu décrit ce que l'analyseur doit s'attendre à trouver comme contenu, et ce qu'il doit en faire quand il le rencontre.

Les deux modèles de contenu que vous trouverez certainement les plus utiles sont `CDATA` et `RCDATA`.

`CDATA` signifie "*Character Data*" - données caractères. Si l'analyseur est à l'intérieur de ce modèle de contenu, il s'attend à trouver des caractères, et uniquement des caractères. Les symboles < et & perdent alors leur signification particulière et sont traités comme de simples caractères.

`RCDATA` signifie "Références à des entités et données caractères". Si l'analyseur est à l'intérieur de ce modèle de contenu, il s'attend à trouver des caractères *et* des entités. < perd sa signification particulière, mais & est toujours compris comme le début d'une entité générale.

C'est particulièrement utile si vous incluez du texte qui contient de nombreux caractères < et &. Vous pourriez bien sûr contrôler que dans votre texte tous les < sont écrits < et tous les & &, il peut être plus facile de marquer la section comme ne contenant que des "CDATA". Quand SGML rencontre l'instruction correspondante, il ignorera les symboles < et & qui apparaîtront dans le contenu.

## Exemple 3.15. Utiliser une section marquée CDATA

```
<para>Voici un exemple de la façon dont vous pourriez inclure
  un texte comportant de nombreux &amp;lt; et &amp;amp;. L'exemple
  lui-même est en HTML. Le texte qui l'encadre (<para> et
  <programlisting>) est du DocBook.</para>

<programlisting>
  <![CDATA[
    <p>Cet exemple vous montre quelques éléments de HTML. ↺
Comme les
      caractères < et > y sont si fréquemment utilisés, il ↺
est plus
      facile de marquer tout l'exemple comme CDATA plutôt que ↺
de se
      servir des entités &amp;agrave; la place de ces caractères ↺
dans tout le
      texte.</p>

    <ul>
      <li>C'est un élément de liste</li>
      <li>C'est un second élément de liste</li>
      <li>C'est un troisième élément de liste</li>
    </ul>
```

```
    <p>C'est la fin de l'exemple.</p>
  -]]>
</programlisting>
```

Si vous consultez le source de ce document, vous verrez qu'il utilise constamment cette technique.

#### 3.8.1.2. `INCLUDE` et `IGNORE`

Si le mot-clé est `INCLUDE`, alors le contenu de la section marquée sera pris en compte. Si le mot-clé est `IGNORE`, alors la section marquée sera ignorée. Il n'apparaîtra pas dans les sorties.

**Exemple 3.16. Utiliser `INCLUDE` et `IGNORE` dans les sections marquées**

```
<![ INCLUDE [
  Ce texte sera traité et inclus.
]]>

<![ IGNORE [
  Ce texte ne sera pas traité ou inclus.
]]>
```

En soi, cela ne sert pas à grand-chose. Si vous vouliez supprimer du texte de votre document, vous auriez pu l'enlever ou le mettre en commentaires.

Cela devient plus utile quand vous comprenez que vous pouvez vous servir des entités paramètres pour contrôler ces sections. Rappelez-vous que les entités paramètres ne peuvent être utilisées que dans un contexte SGML, et une section marquée *est* un contexte SGML.

Si par exemple, vous générez une version imprimée et une version électronique de votre document, vous pourriez vouloir inclure dans la version électronique un contenu supplémentaire qui ne devra pas apparaître dans la version imprimée.

Créez une entité paramètre et donnez lui comme contenu `INCLUDE`. Rédigez votre document en utilisant des sections marquées pour délimiter le contenu qui ne doit apparaître que dans la version électronique. Dans ces sections marquées, servez-vous de l'entité paramètre au lieu du mot-clé.

Lorsque vous voulez générer la version électronique, changez la valeur de l'entité paramètre en `IGNORE` et retraitez le document.

### Exemple 3.17. Utiliser une entité paramètre pour contrôler une section marquée

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % version.electronique "INCLUDE">
]]>

...

<![ %version.electronique [
  Ce texte ne doit apparaître que dans
  la version électronique du document.
]]>
```

Pour générer la version imprimée, changez la définition de l'entité en :

```
<!ENTiTY % version.electronique "IGNORE">
```

A la seconde passe sur le document, les sections marquées qui utilisent `%version.electronique` comme mot-clé seront ignorées.

## 3.8.2. A faire...

1. Créez un nouveau fichier, `section.xml` , qui contienne :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN" [
<!ENTITY % text.output "INCLUDE">
]>

<html>
  <head>
    <title>Exemple d'utilisation des sections marquées</title>
  </head>

  <body>
    <p>Ce paragraphe <![CDATA[contient de nombreux
      caractères < (< < < < <) il est donc
      plus facile de l'inclure dans une section marquée
      CDATA -]]></p>

    <![IGNORE[
```

```
    <p>Ce paragraphe n'apparaîtra jamais dans les
      sorties.</p>
    -]]>

    <![ %sortie.texte [
    <p>Ce paragraphe apparaîtra peut-être dans les
      sorties.</p>

    <p>Cela dépend de l'entité paramètre
      %sortie.texte.</p>
    -]]>
  </body>
</html>
```

2. Normalisez le fichier avec sgmlnorm(1) et examinez le résultat. Notez quels paragraphes ont été conservés et quels paragraphes ont été supprimés, et ce qu'est devenu le contenu des sections marquées CDATA.

3. Modifiez la définition de l'entité `sortie.texte` de `INCLUDE` en `IGNORE`. Normalisez de nouveau le fichier et regardez ce qui a changé dans le résultat.

## 3.9. Conclusion

Ici se termine cette introduction à SGML. Pour des raisons de place et de complexité, de nombreux points ont été survolés (voire omis). Les sections qui précédent décrivent néanmoins suffisamment d'éléments du SGML pour vous permettre de comprendre comment est organisée la documentation du FDP.

# Chapitre 4. Marques SGML

Ce chapitre décrit les trois langages de marquage que vous rencontrerez si vous contribuez au Projet de Documentation de FreeBSD. Chaque section décrit le langage et détaille les marques que vous serez probablement amenés à utiliser, ou qui sont déjà utilisées.

Ces langages sont riches en éléments et il est parfois difficile de savoir lequel employer dans un contexte particulier. Cette section décrit ceux dont vous aurez probablement besoin et donne des exemples de la manière de s'en servir.

Ce n'est *pas* une liste exhaustive d'éléments, cela ne ferait que reprendre le contenu de la documentation de chacun de ces langages. L'objectif de cette section est de lister les éléments qui ont le plus de chance de vous être utiles. Si vous avez des questions sur le type de marque à employer dans un contexte particulier, posez-les s'il vous plaît à la liste de diffusion du Projet de Documentation de FreeBSD, <freebsd-doc@freebsd.org >.

### En ligne vs. de bloc

Dans la suite de ce document, quand on décrira des éléments, *en ligne* signifie que l'élément peut apparaître à l'intérieur d'un bloc et ne génère pas de passage à la ligne. A l'inverse un élément de bloc provoque un passage à la ligne (et d'autres opérations) lorsqu'on le rencontre.

## 4.1. HTML

HTML, l'*HyperText Markup Language* - Langage de Marquage de l'Hypertexte - est le langage de prédilection du World Wide Web. Vous trouverez plus d'informations sur <URL:http://www.w3.org/>.

HTML est utilisé pour marquer les pages du site Web de FreeBSD. Il ne devrait (habituellement) pas servir pour d'autre type de documentation, parce que DocBook offre un jeu de marques beaucoup plus riche. Vous ne devriez donc rencontrez des pages HTML que si vous écrivez pour le site Web.

Il y a eu plusieurs versions de HTML, 1, 2, 3.0, 3.2, et il existe deux variantes de la dernière version, 4.0 (disponible à la fois en version *stricte* et *relâchée*).

Les DTDs HTML existent au catalogue des logiciels portés dans `textproc/html`. Elles sont automatiquement installées par le méta-port `textproc/docproj`.

### 4.1.1. Formal Public Identifier (FPI) - Identifiant Public Formel

Il y a un certain nombre de FPIs HTML, selon la version (qu'on appelle aussi le niveau) de HTML avec laquelle vous voulez que votre document soit compatible.

La plupart des documents HTML du site Web de FreeBSD respectent strictement la version relâchée de HTML 4.0 :

```
PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
```

### 4.1.2. Sections

Un document HTML est habituellement composé de deux sections. La première section, appelée *head* - en-tête, contient des informations sur le document, comme son titre, le nom de son auteur, le document dans lequel il est inclus, et ainsi de suite. La seconde section, le *body* - corps, contient ce qui sera affiché.

Ces sections sont dénotées par les éléments `head` et `body` respectivement. Ces éléments appartiennent à l'élément de premier niveau `html`.

Exemple 4.1. Structure habituelle d'un document HTML

```
<html>
  <head>
   <title>Le titre du document </title>
  </head>

  <body>

    …

  </body>
</html>
```

### 4.1.3. Eléments de blocs

#### 4.1.3.1. Titres

HTML vous permet d'avoir jusqu'à six niveaux de titres différents dans votre document.

Le titre le plus gros et le plus visible est `h1`, puis `h2`, jusqu'à `h6`.

Le contenu de l'élément est le texte du titre.

### Exemple 4.2. `h1`, `h2`, etc.

Utilisez :

```
<h1>Première section</h1>

<!-- Introduction du document -->

<h2>C'est le titre de la première section</h2>

<!-- Contenu de la première section -->

<h3>C'est le titre de la première sous-section</h3>

<!-- Contenu de la première sous-section -->

<h2>C'est le titre de la seconde section</h2>

<!-- Contenu de la seconde section -->
```

Une page HTML doit normalement avoir un titre de premier niveau (`h1`). Il peut contenir plusieurs titres de second niveau (`h2`), et à leur tour, de nombreux titres de troisième niveau. Chaque élément `h`*n* doit appartenir à un même élément de niveau supérieur. Il faut éviter de sauter d'un cran dans la numérotation.

### Exemple 4.3. Mauvais ordonnancement des éléments `h`*n*

Use:

```
<h1>Première section</h1>

<!-- Introduction du document -->

<h3>Sous-section</h3>

<!-- Ce n'est pas bon, <h2> a été oublié -->
```

#### 4.1.3.2. Paragraphes

HTML n'a qu'un seul élément paragraphe, `p`.

Exemple 4.4. `p`

Utilisez :

```
<p>C'est un paragraphe. Il peut contenir pratiquement
  n'importe quel élément.</p>
```

#### 4.1.3.3. Citations

Une citation d'un long extrait d'un autre document, qui ne doit pas apparaître dans le paragraphe en cours, mais est mise dans un bloc de citation.

Exemple 4.5. `blockquote`

Utilisez :

```
<p>Un court extrait de la Constitution des Etats-Unis :</
p>

<blockquote>Nous le Peuple des Etats-Unis, dans le But de ↺
former
  une Union plus parfaite, d'établir la Justice, d'assurer
  la Tranquilité domestique, de défendre chacun, de promouvoir
  le Bien-être général, et de garantir les Bénédictions de
  la Liberté &agrave; nous-mêmes et &agrave; notre Postérité, ↺
décidons et
  établissons cette Constitution des Etats-Unis d'Amérique.</
blockquote>
```

#### 4.1.3.4. Listes

Il y a trois types de listes que vous pouvez afficher : ordonnée, non ordonnée et de définition.

Typiquement, chaque entrée d'une liste ordonnée sera numérotée, alors que chaque entrée d'une liste non ordonnée sera précédée d'une puce. Les listes de définition ont deux

sections pour chaque entrée. La première est le terme que l'on définit et la seconde sa définition.

Les listes ordonnées sont dénotées par l'élément `ol`, les listes non ordonnées par l'élément `ul` et les listes de définition par l'élément `dl` element.

Les listes ordonnées et non ordonnées contiennent des éléments de liste, notés avec l'élément `li`. Un élément de liste peut contenir du texte, ou être décomposé en plusieurs éléments `p`.

Les listes de définition contiennent des termes à définir (`dt`) et leurs définitions (`dd`). Le terme à définir n'est composé que de texte. La définition peut comporter d'autres éléments de blocs.

### Exemple 4.6. `ul` et `ol`

Utilisez :

```
<p>Une liste non ordonnée. Les éléments de la liste seront
  probablement précédés par des puces.</p>

<ul>
  <li>Premier élément</li>

  <li>Second élément</li>

  <li>Troisième élément</li>
</ul>

<p>Une liste ordonnée, dont les éléments comportent plusieurs ↺
paragraphes.
  Chaque élément (note : et non chaque paragraphe) sera ↺
numéroté.</p>

<ol>
  <li><p>C'est le premier élément. Il n'a qu'un paragraphe..</
p></li>

  <li><p>C'est le premier paragraphe du second élément.</p>

    <p>C'est le second paragraphe du second élément.</p>

  <li><p>C'est le premier et seul paragraphe du troisième ↺
élément.</p></li>
</ol>
```

### Exemple 4.7. Listes de définition avec `dl`

Utilisez :

```
<dl>
  <dt>Terme 1</dt>

  <dd><p>Paragraphe 1 de la définition 1.</p></dd>

    <p>Paragraphe 2 de la définition 1.</p></dd>

  <dt>Terme 2</dt>

  <dd><p>Paragraphe 1 de la définition 2.</p></dd>

  <dt>Terme 3</dt>

  <dd>Paragraphe 1 de la définition 3. Remarquez que ↺
l'élément <p> n'est
    pas obligatoire dans le cas d'un paragraphe unique.</dd>
</dl>
```

#### 4.1.3.5. Texte pré-formaté

Vous pouvez préciser que du texte doit apparaître exactement comme il est présenté dans le fichier. Cela signifie habituellement que le texte est affiché en police fixe, que les blancs successifs sont conservés et que les passages à la ligne dans le texte sont significatifs.

Pour cela, il faut mettre ce texte dans un élément `pre`.

### Exemple 4.8. `pre`

Vous pouvez utiliser `pre` pour marquer le texte d'un courrier électronique :

```
<pre>
  From: nik@freebsd.org
  To: freebsd-doc@freebsd.org
  Subject: Nouvelle documentation disponible

  Une nouvelle version de mon introduction pour les nouveaux
  participants au Projet de Documentation de FreeBSD est
```

```
  disponible &agrave; l'adresse suivante :

    <URL:http://www.freebsd.org/~nik/primer/index.html>

  Commentaires souhaités.

  N
</pre>
```

### 4.1.3.6. Tables

#### Note

La plupart des navigateurs en mode texte (comme Lynx) n'affichent pas très bien les tables. Si vous utilisez ce type de présentation en tableaux, vous devriez envisager d'utiliser d'autres marques pour éviter la confusion.

Marquez les tableaux avec l'élément `table`. Un tableau est composé d'une ou plusieurs lignes (`tr`), chacune contenant une ou plusieurs cellules (`td`). Chaque cellule peut contenir d'autres éléments de bloc, des paragraphes ou des listes par exemple. Elle peut aussi contenir d'autres tables (cet emboîtement peut se répéter indéfiniment). Si la cellule ne contient qu'un seul paragraphe, l'élément `p` n'est pas obligatoire.

#### Exemple 4.9. Emploi simple de `table`

Utilisez :

```
<p>C'est une table 2x2 simple.</p>

<table>
  <tr>
    <td>Cellule en haut &agrave; gauche</td>

    <td>Cellule en haut &agrave; droite</td>
  </tr>

  <tr>
    <td>Cellule en bas &agrave; gauche</td>

    <td>Cellule en bas &agrave; droite</td>
```

```
  </tr>
</table>
```

Une cellule peut occuper plusieurs lignes ou colonnes. Pour le préciser, ajoutez les attributs `rowspan` et/ou `colspan`, dont les valeurs donnent le nombre de lignes et de colonnes occupées.

## Exemple 4.10. Emploi de `rowspan`

Utilisez :

```
<p>Une grande cellule &agrave; gauche, deux petites cellule ↺
&agrave; droite.</p>

<table>
  <tr>
    <td rowspan="2">Grande et mince</td>
  </tr>

  <tr>
    <td>Cellule du haut</td>

    <td>Cellule du bas</td>
  </tr>
</table>
```

## Exemple 4.11. Emploi de `colspan`

Utilisez :

```
<p>Une grande cellule en haut, deux petites cellules en ↺
dessous.</p>

<table>
  <tr>
    <td colspan="2">Cellule du haut</td>
  </tr>

  <tr>
    <td>Cellule du bas &agrave; gauche</td>
```

```
    <td>Cellule du bas &agrave; droite</td>
  </tr>
</table>
```

### Exemple 4.12. Emploi de `rowspan` et `colspan` ensemble

Use:

```
<p>Sur une grille 3x3, la cellule en haut &agrave; gauche ↺
s'étend sur deux
  lignes et deux colonnes. Les autres cellules sont normales.</
p>

<table>
  <tr>
    <td colspan="2" rowspan="2">Grande cellule en haut ↺
&agrave; gauche</td>

    <td>Cellule en haut &agrave; droite</td>
  </tr>

  <tr>
    <td>Cellule du milieu &agrave; droite</td>
  </tr>

  <tr>
    <td>Cellule en bas &agrave; gauche</td>

    <td>Cellule en bas au milieu</td>

    <td>Cellule en bas &agrave; droite</td>
  </tr>
</table>
```

## 4.1.4. Eléments

### 4.1.4.1. Information d'accentuation

Il y a deux niveaux d'accentuation disponibles en HTML, `em` et `strong`. `em` marque une accentuation normale et `strong` une accentuation plus prononcée.

`em` est généralement rendu en italiques et `strong` en gras. Ce n'est malgré tout pas toujours le cas, et il ne faut pas se baser là-dessus.

### Exemple 4.13. `em` et `strong`

Utilisez :

```
<p><em>Ceci</em> est accentué, et
  <strong>cela</strong> l'est encore plus.</p>
```

#### 4.1.4.2. Gras et italiques

HTML comporte des marques pour la présentation, vous pouvez donc aussi préciser qu'un contenu donné doit apparaître en gras ou en italiques. Les éléments pour cela sont respectivement `b` et `i`.

### Exemple 4.14. `b` et `i`

```
<p><b>Ceci</b> est en gras, tandis que <i>cela</i> est
  en italiques.</p>
```

#### 4.1.4.3. Texte en police fixe

S'il y a du texte qui doit être affiché en police fixe (machine à écrire), servez-vous de `tt` ( pour “télétype”).

### Exemple 4.15. `tt`

Utilisez :

```
<p>L'auteur original de ce document est
  Nik Clayton, qui peut être contacté par courrier
  électronique &agrave; l'adresse : <tt>nik@freebsd.org</tt>.</
p>
```

#### 4.1.4.4. Taille de police

Vous pouvez préciser qu'un contenu doit être affiché en police plus grande ou plus petite. Il y a trois façons de le faire.

1. Utilisez `big` et `small` pour encadrer le texte dont vous voulez modifier la taille. Ces marques peuvent être imbriquées, il est donc possible d'avoir : `<big><big>C'est bien plus gros</big></big>` .

2. Servez-vous de `font` avec l'attribut `size` prenant respectivement les valeurs `+1` ou `-1`. C'est la même chose que d'utiliser `big` ou `small`. Mais cette façon de faire est obsolète.

3. Utilisez `font` avec l'attribut `size` prenant une valeur de 1 à 7. La taille de police par défaut est 3. Cette façon de faire est aussi obsolète.

Exemple 4.16. **`big`, `small` et `font`**

Les trois extraits suivants ont le même résultat :

```
<p>Ce texte est <small>un peu plus petit</small>.
  Mais celui-l&agrave; <big>est un peu plus gros</big>.</p>

<p>Ce texte est <font size="-1">un peu plus petit</font>.
  Mais celui-l&agrave; <font size="+1">est un peu plus gros</
font>.</p>

<p>Ce texte est <font size="2">un peu plus petit</font>.
  Mais celui-l&agrave; <font size="4">est un peu plus gros</
font>.</p>
```

### 4.1.5. Liens

Note

Les liens font aussi partie du contenu du document.

#### 4.1.5.1. Liens vers d'autres documents sur le WWW

Pour mettre un lien sur un autre document sur le WWW, il faut que vous connaissiez l'URL de ce document.

Ce lien est noté avec `a` et l'attribut `href` contient l'URL du document cible. Le lien est le contenu de l'élément, il est habituellement présenté d'une façon ou d'une autre à l'utilisateur (souligné, couleur différente, curseur de forme différente quand on passe dessus, et ainsi de suite).

### Exemple 4.17. Emploi de `<a href="...">`

Utilisez :

```
<p>Vous trouverez plus d'informations sur le
  <a href="http://www.freebsd.org/">site Web de FreeBSD</a>.</
p>
```

Ces liens amèneront l'utilisateur au début du document sélectionné.

#### 4.1.5.2. Liens sur d'autres parties des documents

Pour mettre un lien sur un endroit précis d'un autre (ou du même) document, il faut que l'auteur de ce document y ait mis des points d'ancrage sur lesquels vous pouvez pointer.

Les points d'ancrage sont notés avec `a` et l'attribut `name` au lieu de `href`.

### Exemple 4.18. Emploi de `<a name="...">`

Utilisez :

```
<p><a name="para1">Ce</a> paragraphe peut être référencé
  par d'autres liens via le nom <tt>para1</tt>.</p>
```

Pour mettre un lien sur une partie nommée d'un document, utilisez un lien ordinaire, mais ajoutez-y le nom du point d'ancrage précédé d'un symbole `#`.

### Exemple 4.19. Lien sur une partie nommée d'un autre document

Supposons que l'exemple `para1` se trouve dans un document appelé `foo.html` .

```
<p>Vous trouverez plus d'informations au
  <a href="foo.html#para1">premier paragraphe</a> de
  <tt>foo.html</tt>.</p>
```

Si le lien pointe sur un point d'ancrage nommé du même document, vous pouvez ommettre son URL et ne mettre que le nom du point d'ancrage (précédé de **#**).

### Exemple 4.20. Lien sur une partie nommée du même document

Supposons que l'exemple `para1` fasse partie de ce document.

```
<p>Vous trouverez plus d'informations au
  <a href="#para1">premier paragraphe</a> de
  ce document.</p>
```

## 4.2. DocBook

DocBook est une DTD créée par le Davenport Group pour la rédaction de documentation technique. De sorte que, et au contraire de LinuxDoc ou HTML, les marques DocBook sont plus conçues pour décrire *ce qu'est* quelque chose que *comment* il faut le présenter.

**`formel` VS. `informel`**

Certains éléments ont deux versions - *formelle* et *informelle*. Habituellement, la version formelle de l'élément comporte une titre. La version informelle n'en a pas.

La DTD DocBook est disponible au catalogue des logiciels portés avec le “méta-logiciel porté” `textproc/docbook`. Elle est automatiquement installée par ce dernier.

### 4.2.1. Extensions FreeBSD

Le Projet de Documentation de FreeBSD a ajouté quelques nouveaux éléments à la DTD DocBook. Ces éléments permettent un marquage plus précis.

Dans la suite, quand il sera question d'un élément propre à FreeBSD, ce sera clairement indiqué.

Le terme “DocBook” désigne dans ce qui suit la DTD DocBook avec les extensions FreeBSD.

### Note

Il n'y a rien dans ces extensions qui soit propre à FreeBSD, on a juste pensé que ce seraient des ajouts utiles pour ce projet précis. Si d'autres contributeurs aux autres projets “*nix” (NetBSD, OpenBSD, Linux, ...) sont intéressés à participer à la mise au point d'un jeu d'extensions DocBook standard, merci de contacter Nik Clayton <nik@FreeBSD.org >.

Les extensions FreeBSD ne font pas (actuellement) partie du catalogue des logiciels portés. Elles sont inclues dans les sources du Projet de Documentation et se trouvent dans `doc/share/xml/freebsd.dtd` .

## 4.2.2. Identifiant Public Formel - Formal Public Identifier, (FPI)

En conformité avec les conventions DocBook concernant les FPIs pour les personnalisations de DocBook, le FPI pour la DTD DocBook avec les extensions FreeBSD est :

```
PUBLIC "-//FreeBSD//DTD DocBook V3.1-Based Extension//EN"
```

## 4.2.3. Structure des documents

DocBook vous permet de structurer votre documentation de différentes façons. Le Projet de Documentation de FreeBSD utilise deux types de documents de base, le livre et l'article.

Un livre est organisé en `chapters`. C'est une obligation. Il peut y avoir des `parts` entre le livre et le chapitre si l'on veut un niveau supplémentaire dans le découpage.

Un chapitre peut avoir (ou non) une ou plusieurs sections. Elles sont désignées par l'élément `sect1`. Si une section inclue une autre section, utilisez l'élément `sect2`, et ainsi de suite, jusqu'à `sect5`.

Le contenu du livre est lui-même dans les chapitres et sections.

Un article est plus simple qu'un livre, et n'a pas de chapitres. Au lieu de cela, le contenu d'un article est organisé en une ou plusieurs sections, à l'aide des mêmes éléments `sect1` (`sect2` et ainsi de suite) dont on se sert pour les livres.

Il vous faudra manifestement choisir le type de document à utiliser selon la nature du document que vous rédigez. Les articles sont bien adaptés pour des documents qui

n'ont pas besoin d'être décomposés en chapitres, et qui sont, relativement parlant, assez court - jusqu'à 20-25 pages. Les livres eux conviennent aux documents qui peuvent être découpés en plusieurs chapitres, avec éventuellement des annexes, et autres composants.

Les guides FreeBSD sont tous des articles, tandis que ce document, la FAQ FreeBSD, et le Manuel de Référence FreeBSD sont des livres.

#### 4.2.3.1. Commencer un livre

Le contenu d'un livre est un inclus dans l'élément `book`. Tout autant que des marques organisant le contenu, cet élément peut contenir des éléments qui donnent des informations supplémentaires sur le livre. Ce sont soit des méta-informations, utilisées pour y faire référence, soit un contenu supplémentaire servant à générer la page de titre.

Ces informations supplémentaires doivent être inclues dans l'élément `bookinfo`.

Exemple 4.21. Boilerplate??? **`book`** avec **`bookinfo`**

```
<book>
  <bookinfo>
    <title>Mettez le titre ici </title>

    <author>
      <firstname>Votre prénom </firstname>
      <surname>Votre nom de famille </surname>
      <affiliation>
        <address><email>Votre adresse de courrier
            électronique</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>1998</year>
      <holder role="mailto:Votre adresse de courrier
          électronique">Votre nom </holder>
    </copyright>

    <pubdate role="rcs">$Date$</pubdate>

    <releaseinfo>$Id$</releaseinfo>

    <abstract>
      <para>Résumez ici le contenu du
          livre.</para>
    </abstract>
  </bookinfo>

  …
```

```
</book>
```

### 4.2.3.2. Commencer un article

Le contenu d'un article est un inclus dans l'élément `article`. Tout autant que des marques organisant le contenu, cet élément peut contenir des éléments qui donnent des informations supplémentaires sur l'article. Ce sont soit des méta-informations, utilisées pour y faire référence, soit un contenu supplémentaire servant à générer la page de titre.

Ces informations supplémentaires doivent être inclues dans l'élément `artheader`.

Exemple 4.22. Boilerplate??? **`article`** avec **`artheader`**

```
<article>
  <artheader>
    <title>Mettez le titre ici </title>

    <author>
      <firstname>Votre prénom </firstname>
      <surname>Votre nom de famille </surname>
      <affiliation>
        <address><email>Votre adresse de courrier
             électronique</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>1998</year>
      <holder role="mailto:Votre adresse de courrier
           électronique">Votre nom </holder>
    </copyright>

    <pubdate role="rcs">$Date$</pubdate>

    <releaseinfo>$Id$</releaseinfo>

    <abstract>
      <para>Résumez ici le contenu de l'article.  </para>
    </abstract>
  </artheader>

  …

</article>
```

#### 4.2.3.3. Séparer les chapitres

Utilisez `chapter` pour marquer vos chapitres. Chaque chapitre a obligatoirement un `title`. Les articles n'ont pas de chapitre, ils sont réservés aux livres.

Exemple 4.23. Un chapitre

```
<chapter>
  <title>Le titre du chapitre</title>

  ...
</chapter>
```

Un chapitre ne peut pas être vide, il doit contenir des éléments en plus du `title`. Si vous voulez inclure un chapitre vide, ajoutez lui simplement un paragraphe vide.

Exemple 4.24. Chapitres vides

```
<chapter>
  <title>C'est un chapitre vide</title>

  <para></para>
</chapter>
```

#### 4.2.3.4. Sections dans les chapitres

Dans les livres, les chapitres peuvent (mais ce n'est pas obligatoire) être décomposés en sections, sous-sections, et ainsi de suite. Dans les articles, les sections sont les principaux éléments d'organisation et chaque article doit contenir au moins une section. Utilisez l'élément `sect`*n*. Le *n* est le type de section, qui indique son niveau de profondeur.

La première `sect`*n* est `sect1`. Vous pouvez en avoir plus d'une dans un chapitre. Elles peuvent inclure un ou plusieurs éléments `sect2`, et ainsi de suite, jusqu'à `sect5`.

Exemple 4.25. Sections dans les chapitres

```
<chapter>
```

```
  <title>Exemple de chapitre</title>

  <para>Du texte dans le chapitre.</para>

  <sect1>
    <title>Première section (1.1)</title>

    &hellip;
  </sect1>

  <sect1>
    <title>Seconde section (1.2)</title>

    <sect2>
      <title>Première sous-section (1.2.1)</title>

      <sect3>
        <title>Première sous-sous-section (1.2.1.1)</title>

        &hellip;
      </sect3>
    </sect2>

    <sect2>
      <title>Seconde sous-section (1.2.2)</title>

      &hellip;
    </sect2>
  </sect1>
</chapter>
```

### Note

Cet exemple donne les numéros des sections dans leurs titres. Vous ne devez pas le faire. Les feuilles de style s'en chargent (voir plus bas pour plus de détails), et vous n'avez pas à vous en préoccupez.

#### 4.2.3.5. Subdivision en `parts`

Vous pouvez avoir un niveau d'organisation supplémentaire entre le `book` et le `chapter` en définissant une ou plusieurs `parts`. Ce n'est pas possible dans un `article`.

```
<part>
  <title>Introduction</title>

  <chapter>
```

```
    <title>Resumé</title>

    ...
  </chapter>

  <chapter>
    <title>Qu'est-ce que FreeBSD ?</title>

    ...
  </chapter>

  <chapter>
    <title>Historique</title>

    ...
  </chapter>
</part>
```

### 4.2.4. Eléments de blocs

#### 4.2.4.1. Paragraphes

DocBook connaît trois types de paragraphes : `formalpara`, `para` et `simpara`.

La plupart du temps, des `para`s vous suffiront. Les `formalpara`s ont un `title`, et avec les `simpara`s, certains éléments sont interdits à l'intérieur du paragraphe. Tenez-vous en aux `para`s.

Exemple 4.26. **para**

Utilisez :

```
<para>C'est un paragraphe. Il peut contenir
  presque n'importe quel autre élément.</para>
```

Apparence :

C'est un paragraphe. Il peut contenir presque n'importe quel autre élément.

#### 4.2.4.2. Citations en bloc

Une citation en bloc est un long extrait d'un autre document qui ne doit pas faire partie du paragraphe courant. Vous n'en aurez probablement pas besoin souvent.

Les citations en blocs peuvent facultativement avoir un titre et une attribution (ou n'avoir ni titre ni attribution).

### Exemple 4.27. `blockquote`

Utilisez :

```
<para>Un court extrait de la constitution des Etats-
Unis :</para>

<blockquote>
  <title>Préambule &agrave; la Constitution des Etats-Unis</
title>

  <attribution>Copié d'un site Web quelque part</attribution>

  <para>Nous le Peuple des Etats-Unis, dans le but de former
    une Union plus parfaite, d'établir la Justice, de garantir
    la Tranquilité domestique, assurer la défense collective,
    promouvoir notre Bien-être général, et conserver &agrave;
    nous-mêmes et &agrave; notre Postérité les bienfaits de la
    Liberté, proclamons et établissons cette Constitution des
    Etats-Unis d'Amérique.</para>
</blockquote>
```

Apparence :

Un court extrait de la constitution des Etats-Unis :

> Préambule à la Constitution des Etats-Unis
>
> Nous le Peuple des Etats-Unis, dans le but de former une Union plus parfaite, d'établir la Justice, de garantir la Tranquilité domestique, assurer la défense collective, promouvoir notre Bien-être général, et conserver à nous-mêmes et à notre Postérité les bienfaits de la Liberté, proclamons et établissons cette Constitution des Etats-Unis d'Amérique.
>
> —Copié d'un site Web quelque part

#### 4.2.4.3. Indications, notes, avertissements, mises en garde, informations importantes et barres verticales.

Vous devrez peut-être ajouter des informations distinctes du texte lui-même. Ce sont habituellement des “méta-informations” que l'utilisateur doit connaître.

Selon la nature de l'information, vous utiliserez l'un des élements `tip`, `note`, `warning`, `caution` ou `important`. Ou bien, si l'information est en rapport avec le texte, mais ne correspond à aucun des cas précédents, servez-vous de `sidebar`.

Les cas où il faut choisir l'un ou l'autre de ces éléments ne sont pas clairement explicités. Voici ce que suggère la documentation DocBook :

- Une Note est une information destinée à tous les lecteurs.
- Un élément Important est une variante de la Note.
- Une *Caution* est une information relative à la perte de données ou dégats logiciels éventuels.
- Un *Warning* est une information relative aux dégats matériels ou risques corporels.

Exemple 4.28. **`warning`**

Utilisez :

```
<warning>
  <para>Installer FreeBSD peut vous donner envie de supprimer
    Windows de votre disque dur.</para>
</warning>
```

Avertissement

Installer FreeBSD peut vous donner envie de supprimer Windows de votre disque dur.

### 4.2.4.4. Listes and procédures

Vous aurez souvent besoin de lister des informations ou d'indiquer à l'utilisateur les différentes étapes nécessaires pour effectuer une tâche donnée.

Pour cela, servez-vous de `itemizedlist`, `orderedlist` ou `procedure`[1]

`itemizedlist` et `orderedlist` sont semblables à leurs contreparties `ul` et `ol` en HTML. Chacune comporte un ou plusieurs éléments `listitem`, et chaque `listitem` contient un ou plusieurs éléments de blocs. Les élements `listitem` sont analogues aux marques `li` du HTML. Néanmoins, au contraire du HTML, ils sont ici obligatoires.

Une `procedure` est légérement différente. Elle consiste en `steps`, qui à leur tour sont composés de `steps` ou `substeps`. Chaque `step` contient des éléments de blocs.

[1] Il y a d'autres types de listes dans DocBook, mais ils ne nous concernent pas pour le moment.

### Exemple 4.29. `itemizedlist` , `orderedlist` et `procedure`

Utilisez :

```
<itemizedlist>
  <listitem>
    <para>C'est le premier élément de la liste.</para>
  </listitem>

  <listitem>
    <para>C'est le second élément de la liste.</para>
  </listitem>
</itemizedlist>

<orderedlist>
  <listitem>
    <para>C'est le premier élément de la liste
      ordonnée.</para>
  </listitem>

  <listitem>
    <para>C'est le second élément de la liste ordonnée.</para>
  </listitem>
</orderedlist>

<procedure>
  <step>
    <para>Faites ceci.</para>
  </step>

  <step>
    <para>Puis cela.</para>
  </step>

  <step>
    <para>Et maintenant cela.</para>
  </step>
</procedure>
```

Apparence :

- C'est le premier élément de la liste.
- C'est le second élément de la liste.

1. C'est le premier élément de la liste ordonnée.

2. C'est le second élément de la liste ordonnée.

1. Faites ceci.

2. Puis cela.

3. Et maintenant cela.

### 4.2.4.5. Extraits de fichiers

Si vous voulez incorporer un extrait de fichier (ou éventuellement une fichier entier), mettez-le dans un élément `programlisting`.

Les blancs et sauts de ligne à l'intérieur de `programlisting` *sont* significatifs. Cela signifie en particulier que la marque de début doit être sur la même ligne que la première ligne du listing et que la marque de fin doit être sur la même ligne que la dernière ligne du listing, sans quoi il y aurait des lignes blanches en trop.

#### Exemple 4.30. `programlisting`

Utilisez :

```
<para>Quand vous aurez fini, votre programme
 ressemblera &agrave; cela :</para>

<programlisting>#include &lt;stdio.h&gt;

int
main(void)
{
    printf("bonjour, le monde\n");
}</programlisting>
```

Notez qu'il faut utiliser les entités correspondantes et non les signes “supérieur” et “inférieur” à la ligne `#include`.

Apparence :

Quand vous aurez fini votre programme, ressemblera à cela :

```
#include <stdio.h>

int
main(void)
```

```
{
    printf("bonjour, le monde\n");
}
```

### Note

DocBook dispose d'un mécanisme pour faire référence à des sections d'un `programlisting` inclus auparavant, appelé “rappels” (voir `programlistingco` pour plus d'information). Je ne le comprend pas tout à fait (i.e., je ne l'ai jamais employé), je ne peux donc pas le décrire. En attendant, vous pouvez numéroter les lignes et y faire référence ensuite. Cela changera dès que je trouverais le temps de comprendre et documenter les “rappels”.

#### 4.2.4.6. Tables

Au contraire d'HTML, vous n'avez pas besoin d'utiliser les tables pour des questions de présentation, puisque les feuilles de style s'en chargent. Les tables servent uniquement pour les données en tableaux.

Brièvement (voyez la documentation de DocBook pour plus de détails), une table (qui peut-être formelle ou informelle) est constituée d'un élément `table`. Il contient au moins un élément `tgroup`, dont l'attribut donne le nombre de colonnes dans ce sous-tableau. Dans le sous-tableau, vous pouvez ensuite avoir un élément `thead`, qui contient les éléments correspondant aux en-têtes des colonnes, et un `tbody` qui contient le corps du tableau.

Les `thead` et `tbody` contiennent des éléments `row` - lignes, qui contiennent à leur tour des éléments `entry`. Chaque élément `entry` correspond à une cellule du tableau.

### Exemple 4.31. `informaltable`

Utilisez :

```
<informaltable>
  <tgroup cols="2">
    <thead>
      <row>
        <entry>C'est le titre de la colonne 1</entry>
        <entry>C'est le titre de la colonne 2</entry>
      </row>
```

```
    </thead>

    <tbody>
      <row>
        <entry>Ligne 1, colonne 1</entry>
        <entry>Ligne 1, colonne 2</entry>
      </row>

      <row>
        <entry>Ligne 2, colonne 1</entry>
        <entry>Ligne 2, colonne 2</entry>
      </row>
    </tbody>
  </tgroup>
</informaltable>
```

Apparence :

| C'est le titre de la colonne 1 | C'est le titre de la colonne 2 |
|---|---|
| Ligne 1, colonne 1 | Ligne 1, colonne 2 |
| Ligne 2, colonne 1 | Ligne 2, colonne 2 |

Si vous ne voulez pas de bordures autour du tableau, vous pouvez ajouter à l'élément `informaltable` l'attribut `frame` avec la valeur `none` (i.e., `<informaltable frame="none">` ).

### Exemple 4.32. Tableaux avec `frame="none"`

Apparence :

| C'est le titre de la colonne 1 | C'est le titre de la colonne 2 |
|---|---|
| Ligne 1, colonne 1 | Ligne 1, colonne 2 |
| Ligne 2, colonne 1 | Ligne 2, colonne 2 |

#### 4.2.4.7. Exemples que l'utilisateur doit suivre

Vous aurez souvent à donner des exemples que l'utilisateur puisse suivre. Ce seront habituellement des dialogues avec la machine : l'utilisateur tape une commande, il reçoit une réponse, il tape une autre commande, et ainsi de suite.

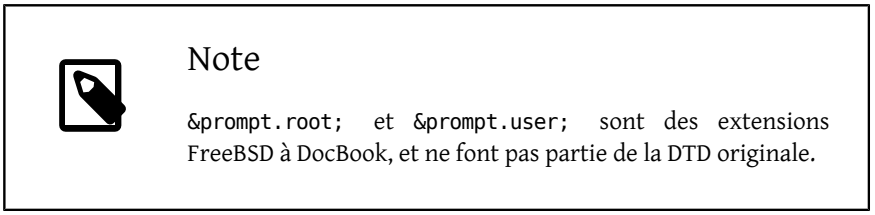
Il y a là un certain nombre d'entités et d'éléments qui entrent en jeu.

`screen`
: Tout ce que l'utilisateur voit dans cet exemple est affiché sur l'écran de l'ordinateur, l'élément suivant est donc `screen`.

  A l'intérieur de `screen`, les blancs sont significatifs.

`prompt`, `&prompt.root;` et `&prompt.user;`
: Parmi ce que l'utilisateur verra à l'écran, il y a les invites (du système, de l'interpréteur de commandes ou des applications). Ils doivent être marqués avec `prompt`.

  Le cas particulier des deux invites de l'interpréteur de commandes, pour un utilisateur ordinaire et pour le super-utilisateur, est traité avec des entités. Chaque fois que vous voulez montrer que l'utilisateur est sous l'interpréteur de commande, servez-vous de `&prompt.root;` ou `&prompt.user;` selon le cas. Il n'y a pas besoin qu'elles soient à l'intérieur de `prompt`.



Note

`&prompt.root;` et `&prompt.user;` sont des extensions FreeBSD à DocBook, et ne font pas partie de la DTD originale.

`userinput`
: Quant il s'agit de texte que l'utilisateur doit taper, mettez-le dans un élément `userinput`. Il sera normalement affiché différement.

Exemple 4.33. **`screen`, `prompt` et `userinput`**

Utilisez :

```
<screen>&prompt.user; <userinput>ls -1</userinput>
foo1
foo2
foo3
&prompt.user; <userinput>ls -1 | grep foo2</userinput>
foo2
&prompt.user; <userinput>su</userinput>
<prompt>Password: </prompt>
&prompt.root; <userinput>cat foo2</userinput>
C'est le fichier 'foo2'</screen>
```

Apparence:

```
% ls -1
foo1
foo2
foo3
% ls -1 | grep foo2
foo2
% su
Password:
# cat foo2
C'est le fichier 'foo2'
```

### Note

Bien que nous montrions le contenu du fichier `foo2`, nous n'utilisons *pas* la marque `programlisting`. Réservez `programlisting` pour les extraits de fichiers hors du dialogue homme/machine.

## 4.2.5. Eléments en ligne

### 4.2.5.1. Mettre de l'information en valeur

Si vous voulez mettre en valeur un mot ou une phrase, servez-vous de `emphasis`. La présentation en sera peut-être en italiques, ou gras, ou bien ce sera prononcé différement par un logiciel de synthèse vocale.

Il n'y a aucun moyen de changer la présentation du texte mis en valeur dans votre document, pas d'équivalent des `b` et `i`. Si l'information que vous donnez est importante, envisagez d'utiliser `important` plutôt que `emphasis`.

### Exemple 4.34. `emphasis`

Utilisez :

```
<para>FreeBSD est sans aucun doute
  <emphasis>le</emphasis> premier système
  d'exploitation de type Unix pour
  architecture Intel.</para>
```

Apparence :

> FreeBSD est sans aucun doute *le* premier système d'exploitation de type Unix pour architecture Intel.

### 4.2.5.2. Applications, commandes, options et références

Il vous arrivera souvent de désigner des applications ou des commandes quand vous rédigerez quelque chose pour le Manuel de Référence. Distinguer les unes des autres est simple : une application est un ensemble de (ou éventuellement un seul) programmes dédiés à une tâche particulière. Une commande est le nom d'un programme que l'utilisateur peut exécuter.

Vous aurez aussi de temps à autre à lister une ou plusieurs des options d'une commande.

Pour finir, il arrivera souvent que vous vouliez indiquer en même temps que la commande, son numéro de section dans les pages de manuel, au format “commande(numéro)” habituel dans les documentations Unix.

Désignez les noms d'applications avec `application`.

Si vous voulez lister une commande avec son numéro de section dans le manuel (ce qui sera la plupart du temps le cas), l'élément DocBook pour cela est `citerefentry`. Il contiendra deux autres éléments, `refentrytitle` et `manvolnum`. Le contenu de `refentrytitle` est le nom de la commande, et celui de `manvolnum` est son numéro de section dans le manuel.

Ce peut être fastidieux à taper, aussi a-t-on défini une séries d'entités générales pour faciliter ces références. Chacune de ces entités est de la forme `&man.page-de-manuel.section-du-manuel;` .

Ces entités sont dans le fichier `doc/share/xml/man-refs.ent` , qui peut-être référencé par le FPI :

```
PUBLIC "-//FreeBSD//ENTITIES DocBook Manual Page Entities//EN"
```

L'introduction de votre documentation ressemblera donc probalement à :

```
<!DOCTYPE book PUBLIC
  "-//FreeBSD//DTD DocBook V3.1-Based Extension//EN" [

<!ENTITY % man PUBLIC
  "-//FreeBSD//ENTITIES DocBook Manual Page Entities//EN">
%man;

…

]>
```

Servez-vous de `command` quand vous voulez mettre le nom d'une commande dans le texte en le présentant comme quelque chose que l'utilisateur doit taper.

Utilisez `option` pour désigner les options d'une commande.

Ce peut être confus, et le bon choix n'est pas toujours évident. Espérons que les exemples qui suivent éclaircirons les choses.

### Exemple 4.35. Applications, commandes et options.

Utilisez :

```
<para><application>Sendmail</application> est le logiciel de
  courrier électronique le plus employé sous Unix.</para>

<para><application>Sendmail</application> comporte les
  programmes <citerefentry>
    <refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum>
  </citerefentry> et &man.newaliases.8;.</para>

<para>L'un des paramètres de la ligne de commande de
  <citerefentry><refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum></citerefentry>,
    <option>-bp</option>, affiche l'état des messages
    dans la file d'attente. Vérifiez-le en exécutant
    <command>sendmail -bp</command>.</para>
```

Apparence :

Sendmail est le logiciel de courrier électronique le plus employé sous Unix.

Sendmail comporte les programmes sendmail(8) et newaliases(8).

L'un des paramètres de la ligne de commande de sendmail(8), `-bp`, affiche l'état des messages dans la file d'attente. Vérifiez-le en exécutant `sendmail -bp`.

#### Note

Remarquez comme il est plus facile d'utiliser la notation `&man.commande.section;` .

#### 4.2.5.3. Fichiers, répertoires, extensions

Chaque fois que vous voulez donner le nom d'un fichier, d'un répertoire ou de l'extension d'un fichier, servez-vous de `filename`.

### Exemple 4.36. `filename`

Utilisez :

```
<para>Vous trouverez le source SGML du Manuel
  de Référence en Anglais dans
  <filename>/usr/doc/en/handbook/</filename>. Le
  fichier principal, dans ce répertoire, s'appelle
  <filename>handbook.xml</filename>. Il doit aussi
  y avoir un <filename>Makefile</filename> et un
  certain nombre de fichiers avec l'extension
  <filename>.ent</filename>.</para>
```

Apparence :

Vous trouverez le source SGML du Manuel de Référence en Anglais dans `/usr/doc/en/handbook/` . Le fichier principal, dans ce répertoire, s'appelle `handbook.xml` . Il doit aussi y avoir un `Makefile` et un certain nombre de fichiers avec l'extension `.ent`.

#### 4.2.5.4. Fichiers spéciaux de périphériques

#### extension FreeBSD

Ces éléments font partie des extensions FreeBSD à DocBook et n'existent pas dans la DTD DocBook d'origine.

Pour faire référence à des fichiers spéciaux de périphériques, vous avez deux solutions. Soit utiliser le nom du fichier spécial dans `/dev`, soit le nom sous lequel il est désigné dans le noyau. Dans ce dernier cas, servez-vous de `devicename`.

Il y a des cas où vous n'aurez pas le choix. Pour certains périphériques, les cartes réseaux par exemple, il n'y a pas d'entrées dans `/dev`, ou bien celles-ci sont très différentes des noms utilisés dans le noyau.

### Exemple 4.37. `devicename`

Utilisez :

```
<para><devicename>sio</devicename> sert
  sous FreeBSD aux communications séries.
  <devicename>sio</devicename> correspond
  &agrave; un certain nombre d'entrées dans
  <filename>/dev</filename>, dont
  <filename>/dev/ttyd0</filename> et
  <filename>/dev/cuaa0</filename>.</para>

<para>A l'inverse, les périphériques réseaux, tel que
  <devicename>ed0</devicename> n'apparaissent pas dans
  <filename>/dev</filename>.</para>

<para>Sous MS-DOS, le premier lecteur de disquette s'appelle
  <devicename>a:</devicename>. Sous FreeBSD, c'est
  <filename>/dev/fd0</filename>.</para>
```

Apparence :

`sio` sert sous FreeBSD aux communications séries. `sio` correspond à un certain nombre d'entrées dans `/dev`, dont `/dev/ttyd0` et `/dev/cuaa0` .

A l'inverse, les périphériques réseaux, tel que `ed0` n'apparaissent pas dans `/dev`.

Sous MS-DOS, le premier lecteur de disquette s'appelle `a:`. Sous FreeBSD, c'est `/dev/fd0` .

### 4.2.5.5. Machines, domaines, adresses IP, et ainsi de suite

#### extension FreeBSD

Ces éléments font partie des extensions FreeBSD à DocBook et n'existent pas dans la DTD DocBook d'origine.

Il y a différentes façons de dénoter les informations d'identification des machines en réseau, selon le type d'information. Elles utilisent toutes `hostid` comme élément, l'attribut `role` servant à qualifier le type d'information.

Pas de valeur de l'attribut, ou `role="hostname"`
: Sans valeur de l'attribut (i.e., `hostid` ...`hostid` ), l'information donnée est le seul nom de la machine, `freefall` ou `wcarchive`, par exemple. Vous pouvez l'expliciter avec `role="hostname"` .

`role="domainname"`
: C'est ici un nom de domaine, comme `FreeBSD.org` ou `ngo.org.uk` . Il n'y a pas de nom de machine.

`role="fqdn"`
: C'est le nom complet de la machine - *Fully Qualified Domain Name*, composé du nom de la machine et du nom de domaine.

`role="ipaddr"`
: On donne alors l'adresse IP, probablement sous forme de quatre nombres séparés par des points.

`role="netmask"`
: C'est un masque de sous-réseau, qui peut être donné comme quatre nombres séparés par des points, un chaîne en hexadécimal ou un `/` suivi d'un nombre.

`role="mac"`
: C'est une adresse Ethernet MAC, exprimée par un séries de valeurs hexadécimales sur deux positions séparées par des deux-points.

## Exemple 4.38. `hostid` et `role`s

Utilisez :

```
<para>La machine locale peut toujours
  être désignée par <hostid>localhost</hostid>,
  et aura l'adresse IP
  <hostid role="ipaddr">127.0.0.1</hostid>.</para>

<para>Le domaine
  <hostid role="domainname">FreeBSD.org</hostid>
  inclut un certain nombre de machine, dont
  <hostid role="fqdn">freefall.FreeBSD.org</hostid> et
  <hostid role="fqdn">bento.FreeBSD.org</hostid>.</para>

<para>Quand vous ajoutez un alias IP &agrave; une interface ↺
(avec
  <command>ifconfig</command>), utilisez
  <emphasis>toujours</emphasis> le masque de sous-réseau
  <hostid role="netmask">255.255.255.255</hostid>
  (qui peut aussi s'écrire
  <hostid role="netmask">0xffffffff)</hostid>.</para>

<para>L'adresse MAC identifie de façon unique
  chaque carte réseau existante.  Une adresse MAC
  ressemble typiquement &agrave; <hostid
    role="mac">08:00:20:87:ef:d0</hostid>.</para>
```

Apparence :

La machine locale peut toujours être désignée par `localhost`, et aura l'adresse IP `127.0.0.1`.

Le domaine `FreeBSD.org` inclut un certain nombre de machine, dont `freefall.FreeBSD.org` et `bento.FreeBSD.org` .

Quand vous ajoutez un alias IP à une interface (avec `ifconfig`), utilisez *toujours* le masque de sous-réseau `255.255.255.255` (qui peut aussi s'écrire `0xffffffff` ).

L'adresse MAC identifie de façon unique chaque carte réseau existante. Une adresse MAC ressemble typiquement à `08:00:20:87:ef:d0` .

#### 4.2.5.6. Noms d'utilisateurs

extension FreeBSD

Ces éléments font partie des extensions FreeBSD à DocBook et n'existent pas dans la DTD DocBook d'origine.

Si vous avez besoin de faire référence à un nom d'utilisateur, comme `root` ou `bin`, servez-vous de `username`.

Exemple 4.39. **`username`**

Utilisez :

```
<para>Pour effectuer la plupart des
  tâches d'administration système,
  vous aurez besoin d'être
  <username>root</username>.</para>
```

Apparence :

Pour effectuer la plupart des tâches d'administration système, vous aurez besoin d'être `root`.

### 4.2.5.7. Décrire les `Makefiles`

extension FreeBSD

Ces éléments font partie des extensions FreeBSD à DocBook et n'existent pas dans la DTD DocBook d'origine.

Il y a des éléments pour décrire les composantes d'un `Makefiles`, `maketarget` et `makevar`.

`maketarget` désigne une cible définie dans un `Makefile` qui peut être utilisée en paramètre de `make`. `makevar` désigne une variable qui peut être définie (dans l'environnement, sur la ligne de commande de `make` ou dans le `Makefile`) et affecte le processus .

Exemple 4.40. **`maketarget`** et **`makevar`**

Utilisez :

```
<para>Il y a deux cibles courantes dans les
  <filename>Makefile</filename> :
  <maketarget>all</maketarget> et
  <maketarget>clean</maketarget>.</para>

<para>Généralement, invoquer <maketarget>all</maketarget>
  reconstruit l'application, et invoquer
  <maketarget>clean</maketarget> supprime les
  fichiers temporaires (<filename>.o</filename>
  par exemple) créés lors de la reconstruction.</para>

<para><maketarget>clean</maketarget> peut être
  contrôlée par un certain nombre
  de variables, dont <makevar>CLOBBER</makevar> et
  <makevar>RECURSE</makevar>.</para>
```

Apparence :

Il y a deux cibles courantes dans les `Makefile` : `all` et `clean`.

Généralement, invoquer `all` reconstruit l'application, et invoquer `clean` supprime les fichiers temporaires (`.o` par exemple) créés lors de la reconstruction.

`clean` peut être contrôlée par un certain nombre de variables, dont `CLOBBER` et `RECURSE`.

#### 4.2.5.8. Litéraux

Vous aurez souvent besoin d'inclure “litéralement” du texte dans le Manuel. Ce sont des extraits d'un fichier, que l'on doit pouvoir copier tel quel dans un autre fichier.

Il vous suffira parfois de `programlisting` pour cela. `programlisting` n'est pas toujours approprié, en particulier quand vous voulez inclure un extrait de fichier au fil de l'eau, dans le corps même du paragraphe.

Servez-vous dans ces cas-là de `literal`.

Exemple 4.41. **`literal`**

Utilisez :

```
<para>La ligne <literal>maxusers 10</literal> du fichier
  de configuration du noyau détermine la table de nombreuses
  tables système et définit approximativement le nombre de
  connexions simultanées qu'acceptera le système.</para>
```

Apparence :

La ligne `maxusers 10` du fichier de configuration du noyau détermine la table de nombreuses tables système et définit approximativement le nombre de connexions simultanées qu'acceptera le système.

#### 4.2.5.9. Montrer ce que l'utilisateur doit renseigner

Il arrivera souvent que vous vouliez montrer à l'utilisateur ce qu'il doit faire, faire référence à un fichier, à une ligne de commande, ou autre, dans lesquels l'utilisateur ne pourra pas purement et simplement copier les examples que vous lui donnez, mais devra y renseigner lui-même certaines informations.

`replaceable` est prévu pour ces cas-là. Servez-vous en *à l'intérieur* d'autres éléments pour indiquer quels contenus l'utilisateur doit remplacer.

Exemple 4.42. **`replaceable`**

Utilisez :

```
<informalexample>
  <screen>&prompt.user; <userinput>man
  <replaceable>command</replaceable></userinput></screen>
</informalexample>
```

Apparence :

```
% man command
```

`replaceable` peut servir dans de nombreux autres éléments, dont `literal`. Cet exemple montre aussi qu'il ne faut mettre `replaceable` qu'autour du contenu que l'utilisateur *doit* fournir. Il faut laisser le reste tel quel.

Utilisez :

```
<para>La ligne <literal>maxusers 10</literal> du fichier
  de configuration du noyau détermine la table de nombreuses
  tables système et définit approximativement le nombre
  de connexions simultanées qu'acceptera le système.</para>

<para><literal>32</literal> est un valeur correcte de
  <replaceable>n</replaceable> pour une station
  de travail.</para>
```

Apparence :

La ligne `maxusers 10` du fichier de configuration du noyau détermine la table de nombreuses tables système et définit approximativement le nombre de connexions simultanées qu'acceptera le système.

`32` est un valeur correcte de *n* pour une station de travail.

## 4.2.6. Liens

### Note

Les liens sont aussi des éléments en ligne.

### 4.2.6.1. Mettre des liens vers d'autres parties du même document

Pour mettre de liens à l'intérieur même du document, il faut que vous précisiez d'où part le lien (i.e., le texte ou autre, sur lequel l'utilisateur clique) et où il va.

Chaque élément DocBook possède un attribut `id`. Vous pouvez utiliser cet attribut pour donner un nom unique à l'élément.

C'est cette valeur que vous utiliserez quand vous préciserez la destination du lien.

Habituellement, vous mettrez des liens sur des chapitres ou des sections, vous ajouterez donc un attribut `id` à ces éléments.

### Exemple 4.43. `id` de chapitres et de section

```
<chapter id="chapitre1">
  <title>Introduction</title>

  <para>C'est l'introduction. Elle comporte une sous-section,
    qui a aussi un identifiant.</para>

  <sect1 id="chapter1-sect1">
    <title>Sous-section 1</title>

    <para>C'est la sous-section.</para>
  </sect1>
</chapter>
```

Vous devriez utiliser des valeurs plus explicites. Ces valeurs doivent être uniques pour le document (i.e., pas uniquement dans le fichier, mais dans le document dans lequel le fichier peut éventuellement être inclus aussi). Remarquez que l'`id` de la sous-section est construit en le préfixant de l'`id` du chapitre. Cela aide à construire des identifiants uniques.

Si vous voulez que l'utilisateur puisse aller à un endroit précis du document (éventuellement au milieu du paragraphe), ou à un exemple, servez-vous de `anchor`. Cet élément n'a pas de contenu, mais il a un attribut `id`.

### Exemple 4.44. `anchor`

```
<para>Ce paragraphe inclut un
  <anchor id="para1">lien interne. Il n'apparaît
  pas dans le document.</para>
```

Si vous voulez fournir à l'utilisateur un lien qu'il puisse activer (probablement en cliquant dessus) pour aller à une section du document qui a un attribut `id`, vous pouvez vous servir de `xref` ou `link`.

Ces deux éléments ont un attribut `linkend`. La valeur de cette attribut doit être celle que vous avez utilisée comme attribut `id` (peu importe si cette valeur n'a pas encore été définie dans le document, les liens peuvent être en avant ou en arrière).

Si vous vous servez de `xref`, vous n'avez pas le contrôle du texte du lien. Il sera généré automatiquement.

### Exemple 4.45. Se servir de `xref`

Supposons que ce fragment apparaisse quelque part dans un document qui contienne l'exemple que nous avons donné pour `id` :

```
<para>Vous trouverez plus d'information
  au <xref linkend="chapter1">.</para>

<para>Vous trouverez des informations plus détaillées dans
  <xref linkend="chapter1-sect1">.</para>
```

Le texte du lien sera généré automatiquement, et cela ressemblera à (le texte mis *en valeur* indique que c'est cela le lien) :

> Vous trouverez plus d'information au *Chapitre Un*.
>
> Vous trouverez des informations plus détaillées dans *la section appellée Sous-section 1*.

Remarquez que le texte du lien est construit à partir du titre de la section ou du numéro du chapitre.

### Note

Cela veut dire que vous *ne pouvez pas* utiliser `xref` pour mettre un lien sur l'attribut `id` d'un élément `anchor`. L'`anchor` n'a pas de contenu et `xref` ne pourrait donc pas générer le texte du lien.

Si vous voulez avoir la maîtrise du texte du lien, servez-vous alors de `link`. Cet élément encadre un contenu qui sera utilisé comme lien.

### Exemple 4.46. Utiliser `link`

Supposons que ce fragment apparaisse quelque part dans un document qui contienne l'exemple que nous avons donné pour `id` :

```
<para>Vous trouverez plus d'information
  au <link linkend="chapter1">premier chapitre</link>.</para>

<para>Vous trouverez des informations plus détaillées dans
  <link linkend="chapter1-sect1">cette</link>
  section.</para>
```

Ce qui générera (le texte mis *en valeur* indique que c'est cela le lien) :

> Vous trouverez plus d'information au *premier chapitre*.
>
> Vous trouverez des informations plus détaillées dans *cette* section.

### Note

Le dernier exemple n'est pas à suivre. N'utilisez jamais “ce” ou “ici” comme origine du lien. Le lecteur devra détailler le contexte dans lequel c'est employé pour comprendre où le lien va le mener.

### Note

Vous *pouvez* vous servir de `link` pour mettre un lien sur un `id` ou une `anchor`, puisque le contenu du `link` définit le texte qui sera utilisé comme lien.

#### 4.2.6.2. Liens vers d'autres documents sur le WWW

Mettre des liens sur des documents externes est beaucoup plus facile, si tant est que vous connaissiez l'URL du document sur lequel vous voulez mettre un lien. Servez-vous de `ulink`. L'attribut `url` sera l'URL de la page où pointera le lien, et le contenu du lien sera utilisé pour que l'utilisateur puisse l'activer.

### Exemple 4.47. `ulink`

Utilisez :

```
<para>Vous pouvez bien sûr cessez de lire
```

```
  ce document, et aller au lieu de cela sur la <ulink
    url="http://www.FreeBSD.org/"> page Web de FreeBSD</
ulink>.</para>
```

Apparence :

Vous pouvez bien sûr cessez de lire ce document, et aller au lieu de cela sur la page Web de FreeBSD.

## 4.3. * LinuxDoc

LinuxDoc est adapté de la DTD QWERTZ, et a été d'abord utilisé par le Projet de Documentation de Linux, puis adopté ensuite par celui de FreeBSD.

La DTD LinuxDoc utilise des marques qui décrivent avant tout l'apparence du document et non son contenu (i.e., elle décrit à quoi quelque chose ressemble, et non ce que c'est).

Et le Projet de Documentation de FreeBSD et celui de Linux sont en train de migrer de la DTD LinuxDoc à la DTD DocBook.

La DTD LinuxDoc DTD est disponible au catalogue des logiciels portés, dans la catégorie `textproc/linuxdoc`.

# Chapitre 5. * Feuilles de style

SGML ne décrit pas comment un document doit être affiché ou imprimé. A cet effet, différents langages ont été conçus pour définir des feuilles de style, dont DynaText, Panorama, SPICE, JSSS, FOSI, CSS, et DSSSL.

Pour DocBook, nous utilisons des feuilles de style écrites en DSSSL. Pour le HTML, nous utilisons CSS.

## 5.1. * DSSSL

Le Projet de Documentation utilise une version un minimum personnalisée des feuilles de style DocBook modulaires de Norm Walsh.

Vous les trouverez dans `textproc/dsssl-docbook-modular` .

## 5.2. * CSS

# Chapitre 6. * La Foire Aux Questions

# Chapitre 7. * Le Manuel de Référence

## 7.1. Organisation logique

Le Manuel de Référence est rédigé en conformité avec la DTD DocBook étendue de FreeBSD.

Le Manuel de Référence est un `book` DocBook. Il est ensuite divisé en `parts`, qui contiennent elles-mêmes plusieurs `chapters`. Les `chapters` sont eux-mêmes composés de sections (`sect1`) et sous-sections (`sect2`, `sect3`) et ainsi de suite.

## 7.2. Organisation physique

Le Manuel de Référence (et ses traductions) sont dans le sous-répertoire `doc/langue/handbook` des archives CVS principales. *`langue`* est le code ISO pour la langue, `en`, pour l'Anglais, `ja` pour le Japonais, et ainsi de suite.

Il y a un certain nombre de fichiers et répertoires dans le répertoire `handbook`.

Note

L'organisation du Manuel de Référence sera peut-être modifiée avec le temps, et le présent document peut ne pas être en phase avec ces changements. Si vous avez des questions sur la façon dont le Manuel de Référence est organisé, contactez s'il vous plaît le Projet de Documentation de FreeBSD, <doc@FreeBSD.ORG >.

### 7.2.1. `Makefile`

Le `Makefile` décrit les règles utilisées pour convertir le Manuel de Référence à partir du source (DocBook) dans plusieurs formats cibles (dont HTML, PostScript, et texte).

Le `Makefile` est décrit plus en détail à la Section 7.4, « * Conversion du Manuel dans d'autres formats ».

### 7.2.2. handbook.xml

C'est la racine du Manuel de Référence. Il contient la déclaration DOCTYPE du Manuel, ainsi que les éléments qui décrivent sa structure.

handbook.xml utilise des entités paramètres pour charger les fichiers d'extensions .ent. Ces fichiers (décrits plus loin) définissent à leur tour des entités générales qui sont elles-mêmes employées dans le reste du Manuel.

### 7.2.3. répertoire/chapter.xml

Chaque chapitre du manuel est composé d'un fichier chapter.xml dans un sous-répertoire séparé pour chaque chapitre. Chaque répertoire prend le nom de l'attribut id de l'élément chapter.

Si par exemple, un des chapitres contient :

```
<chapter id="kernelconfiguration">
...
</chapter>
```

il s'appelera alors chapter.xml dans le répertoire kernelconfiguration. Habituellement, il y aura un seul fichier pour tout le chapitre.

A la génération de la version HTML du Manuel, on obtiendra un kernelconfiguration.html . C'est dû à la valeur du id, et non au nom du répertoire.

Dans les versions précédentes du Manuel, ces fichiers étaient dans le même répertoire que handbook.xml , avec le même nom que l'attribut id de l'élément chapter du fichier. Ils ont été déplacés dans des répertoires séparés en prévision des évolutions à venir du Manuel. Il sera en particulier bientôt possible d'inclure des images dans chaque chapitre. Il est donc plus logique que celles-ci soient rangées dans un répertoire où se trouve aussi le texte du chapitre plutôt que de mettre le texte de chaque chapitre, et donc toutes les images dans un même répertoire. Il y aurait fatalement des conflits de nom, alors qu'il est plus facile de travailler avec plusieurs répertoires contenant quelques fichiers qu'avec un seul répertoire dans lequel il y a beaucoup de fichiers.

Un bref coup d'oeil montre qu'il y a de nombreux répertoires avec chacun un fichier chapter.xml dont basics/chapter.xml , introduction/chapter.xml et printing/chapter.xml .

Important

Les noms des chapitres et/ou répertoires ne doivent pas faire référence à leur place dans le Manuel. Cet ordre peut changer quand

> le contenu du Manuel est réorganisé ; ce type de réorganisation ne devrait (normalement) pas entraîner de modification des noms des fichiers (à moins que des chapitres entiers ne changent de niveau dans la hiérarchie).

Chaque fichier `chapter.xml` n'est pas un document SGML intégral. Ils n'ont en particulier pas de déclaration DOCTYPE en tête du fichier.

C'est dommage pour deux raisons :

- Il n'est pas possible de les traiter comme des fichiers SGML et de les convertir en HTML, RTF, PS et autres formats de la même manière que le Manuel. Cela vous *oblige* à recompiler tout le Manuel chaque fois que vous voulez vérifier les effets d'une modification d'un seul chapitre.
- Le `sgml-mode` d'Emacs ne peut pas s'en servir pour déterminer quelle DTD utiliser, ce qui fait perdre les bénéfices de fonctionnalités utiles du `sgml-mode` (complétion des éléments, validation automatique, et ainsi de suite).

## 7.3. Guide de style

Respectez s'il vous plaît les conventions de style ci-dessous pour garder aux sources du Manuel une consistance malgré les nombreuses personnes qui interviennent dessus.

### 7.3.1. Majuscules et minuscules

Les marques doivent être en minuscules `<para>` et *non* `<PARA>`.

Le texte dans les contextes SGML est normalement en majuscules, `<!ENTITY…>` ou `<!DOCTYPE…>` et *non* `<!entity…>` ou `<!doctype…>`.

### 7.3.2. Indentation

Chaque fichier est indenté à partir de la colonne 0, *quel que soit* le niveau d'indentation dans le fichier où il est inclus.

Chaque marque de début augmente l'indentation de 2 blancs et chaque marque de fin la décrémente d'autant. Le contenu des éléments doit être indenté de 2 blancs s'il court sur plusieurs lignes.

A titre d'exemple, voici à quoi ressemble le source de cette section :

```
+--- C'est la colonne 0
```

```
<chapter>
  <title>...</title>

  <sect1>
    <title>...</title>

    <sect2>
      <title>Indentation</title>

      <para>Chaque fichier est indenté &agrave; partir de la ↺
colonne 0,
 <emphasis>quel que soit</emphasis> le niveau d'indentation dans le
        fichier où il est inclus.</para>

      <para>Chaque marque de début augmente l'indentation de 2 ↺
blancs et
        chaque marque de fin la décrémente d'autant. Le contenu des ↺
éléments
        doit être indenté de 2 blancs s'il court sur plusieurs ↺
lignes.</para>

      ...
    </sect2>
  </sect1>
</chapter>
```

Si vous vous servez d'Emacs ou Xemacs pour éditer les fichiers, le `sgml-mode` doit être chargé automatiquement, et les variables Emacs locales en fin de chaque fichier doivent mettre ces indentations en pratique.

### 7.3.3. Modifications de présentation des sources

Quand vous intégrez des modifications, *ne faites pas en même temps de modification de contenu et de présentation des sources.*

Cela pour que les équipes de traductions du Manuel puissent rapidement voir les modifications de contenu que vous avez intégrées, sans avoir à se demander si une ligne a changé de contenu ou simplement de présentation.

Si vous avez par exemple ajouté deux phrases à un paragraphe, de sorte que les lignes du paragraphe débordent maintenant des 80 colonnes, intégrez d'abord la modification avec les lignes trop longues. Puis corrigez ensuite ce problème, en précisant qu'il ne s'agit que d'une modification de présentation, dont les équipes de traduction n'ont pas à se soucier.

## 7.4. * Conversion du Manuel dans d'autres formats

# Chapitre 8. * Le site Web

# Chapitre 9. Traductions

Ceci est la Foire aux Questions pour les gens qui traduisent la documentation de FreeBSD (FAQ, Manuel de Référence, guides, pages de manuel et autres) en différentes langues.

Elle est *très* largement basée sur la FAQ du Projet Allemand de Documentation de FreeBSD, rédigée à l'origine par Frank Grnder <elwood@mc5sys.in-berlin.de > et traduite en Anglais par Bernd Warken <bwarken@mayn.de >.

Nik Clayton <nik@FreeBSD.org > maintient cette FAQ.

Q : Pourquoi une FAQ ?

R : De plus en plus de gens s'adressent à la liste de diffusion du groupe de documentation de FreeBSD et se proposent de traduire en d'autres langues la documentation de FreeBSD. Le but de cette FAQ est de répondre à leurs questions de façon à ce qu'ils puissent commencer le plus rapidement possible à traduire la documentation.

Q : Que signfient i18n et l10n ?

R : i18n veut dire internationalisation et l10n localisation. Ce ne sont que des abréviations commodes.

i18n se lit “i” suivi de 18 lettres, suivies d'un “n”. De la même façon, l10n se lit “l” suivi de 10 lettres, suivies d'un “n”.

Q : Y-a-t-il une liste de diffusion pour les traducteurs ?

R : Oui, <freebsd-translate@ngo.org.uk >. Inscrivez-vous en envoyant un message à <freebsd-translate-request@ngo.org.uk > avec le mot `subscribe` dans le texte du message.

Vous recevrez une réponse vous demandant de confirmer votre inscription (de la même façon que pour les listes de FreeBSD sur `FreeBSD.org` ).

La langue de base sur la liste est l'Anglais, mais les messages en d'autres langues sont acceptés. La liste n'est pas modérée, mais vous devez vous y être inscrit avant d'y adresser des messages.

La liste est archivée, mais il n'est pas actuellement possible d'y faire des recherches. Vous aurez en retour des explications sur la façon d'accéder aux archives en envoyant le message `help` à <majordomo@ngo.org.uk >.

Il est prévue que la liste de diffusion soit transférée sur `FreeBSD.org` et devienne donc *officielle* dans un avenir proche.

Q : A-t-on besoin de nouveaux traducteurs ?

R : Oui. Plus il y a de gens qui travaillent aux traductions, plus vite elles sont réalisées, et synchronisées avec les modifications de la version anglaise.

Il n'est pas nécessaire que vous soyez traducteur professionnel pour pouvoir contribuer.

Q : Quelle langue faut-il que je connaisse ?

R : Dans l'idéal, il faudrait que vous ayez une bonne connaissance de l'anglais écrit et bien sûr, il faut que vous pratiquiez couramment la langue dans laquelle vous traduisez.

L'anglais n'est pas strictement indispensable. Vous pourriez par exemple effectuez une traduction en Hongrois, à partir de la traduction espagnole de la FAQ.

Q : Quels logiciels faut-il que je connaisse ?

R : Il est fortement recommandé que vous teniez à jour une copie locale des archives CVS de FreeBSD (au moins de la documentation), soit à l'aide de CTM, soit en utilisant CVSup. Le chapitre “Se synchroniser avec la version `-current` de FreeBSD” du Manuel de Référence vous explique comment utiliser ces logiciels.

Il faudra que vous soyiez à l'aise avec CVS. Cela vous permettra de suivre les modifications apportées entre les différentes versions des fichiers qui constituent la documentation.

[A Faire -- écrire un mode d'emploi qui explique comment utiliser CVSup pour ne récupérer que la documentation, et voir ce qui a évolué entre deux versions données]

Q : Comment savoir qui d'autre traduit la documentation dans la même langue ?

R : La page des traductions du Projet de Documentation liste les traductions en cours déjà connues. S'il y a déjà d'autres personnes qui travaillent à la traduction de documentation dans votre langue, s'il vous plaît, ne faites pas le même travail qu'eux en double. Au lieu de cela, prenez contact avec eux, pour savoir comment vous pouvez les aider.

S'il la page n'indique personne qui travaille dans votre langue, envoyez un message à la liste de diffusion du groupe de documentation de FreeBSD au cas où quelqu'un aurait déjà envisagé de commencer une traduction, mais que ne ce soit pas encore signalé.

Q : Il n'y a personne d'autre qui traduise dans ma langue. Que faire ?

R : Félicitations, vous venez juste de lancer le “Projet de Documentation *`dans-votre-langue`* de FreeBSD. Bien venu à bord.

Commencez par vous assurer que vous avez bien du temps disponible. Comme vous êtes pour le moment le seul volontaire pour la traduction dans votre langue, il se-

ra de votre responsabilité de publier votre travail et de coordonner celui d'autres personnes qui voudraient y collaborer.

Envoyez un courrier électronique à la liste de diffusion du groupe de documentation de FreeBSD pour annoncer que vous allez traduire la documentation de façon à ce que la pages des traductions du Projet de Documentation puisse être tenue à jour.

Il faudra vous inscrire à la liste de diffusion <freebsd-translate@ngo.org.uk > (comme expliqué plus haut).

S'il y a déjà dans votre pays quelqu'un qui maintienne un site miroir de FreeBSD, vous devriez le contacter et voir s'il peut vous fournir un hébergement pour votre projet et, si possible, une adresse de courrier électronique et la possibilité de mettre en place une liste de diffusion.

Choisissez ensuite une documentation et commencez la traduction. Il vaut mieux commencer par quelque chose de pas trop volumineux—soit la FAQ, soit l'un des guides.

Q : J'ai traduit une documentation, à qui dois-je l'envoyer ?

R : Cela dépend. Si vous travaillez déjà au sein d'une équipe de traduction (comme l'équipe Japonaise ou l'équipe Allemande), elle aura ses propres procédures pour gérer la documentation soumise, qui seront décrites dans ses pages Web.

Si vous êtes la seule personne à travailler dans une langue donnée (ou si vous êtes responsable d'un projet de traduction et voulez soumettre votre travail au Projet FreeBSD), vous devez alors l'envoyer au Projet FreeBSD (voir la question suivante).

Q : Je suis la seule personne à traduire dans cette langue, comment soumettre mes traductions ?

ou

Nous sommes une équipe de traduction et voulons soumettre les traductions de nos membres ?

R : Commencez par vérifier que vos traductions sont correctement structurées. C'est-à-dire qu'elles doivent s'intégrer à l'arborescence des documentations existantes et compiler sans problèmes.

La documentation de FreeBSD est actuellement archivée dans les répertoires en dessous de `doc/`. Les sous-répertoires de celui-ci prennent le nom codifiant la langue dans laquelle les documentations sont écrites, selon la norme ISO639 (`/usr/share/misc/iso639` , pour les versions de FreeBSD postérieures au 20 Janvier 1999).

S'il ya a différentes codifications pour votre langue (le Chinois,par exemple), il y aura au niveau en-dessous plusieurs sous-répertoires, un pour chacun des formats de codage que vous aurez fournis.

Vous aurez enfin des sous-répertoires pour chaque document.

Une éventuelle traduction en Suédois ressemblerait par exemple à ce qui suit :

```
      doc/
          sv/
      Makefile
      FAQ/
          Makefile
   *.xml
```

`sv` est le code ISO639 pour le Suédois. Remarquez les deux `Makefiles`, qui servent à compiler la documentation. Il n'y a qu'une seule façon de coder le Suédois, il n'y a donc pas de niveau intermédiaire entre les répertoires `sv` et `FAQ`[1].

Utilisez tar(1) et gzip(1) pour compresser votre documentation, et envoyez-la au projet.

```
% cd doc
% tar cf swedish-docs.tar sv
% gzip -9 swedish-docs.tar
```

Mettez `swedish-docs.tar.gz` quelque part. Si vous ne disposez pas d'espace sur le Web (votre fournisseur d'accès n'en met peut-être pas à votre disposition), vous pouvez envoyer un courrier électronique à Nik Clayton, pour vous mettre d'accord sur une date pour les lui envoyer par courrier.

De quelque façon que vous procédiez, il faudra que vous utilisez send-pr(1) pour envoyer un rapport signalant que vous avez soumis de la documentation. Il serait très utile que d'autres puissent relire votre traduction d'abord, parce qu'il y a peu de chances que la personne qui l'intégrera connaisse bien votre langue.

Quelqu'un (probablement le Responsbale du Projet de Documentation, Nik Clayton actuellement), récupérera votre traduction et s'assurera qu'elle compile. En particulier, les points suivants seront examinés :

1. Tous vos fichiers utilisent-ils de chaînes RCS (comme “ID”).

2. `make all` fonctionne-t-il correctement dans le répertoire `sv`.

3. `make install` fonctionne-t-il correctement.

S'il y a des problèmes, la personne qui examine votre soumission vous en fera part pour que vous essayiez de les régler.

S'il n'y a pas de problèmes, votre traduction sera intégrée aussi vite que possible.

[1]Cette organisation va radicalement changer très bientôt. Suivez ce qui ce dit sur la liste de diffusion du groupe de documentation de FreeBSD pour plus d'information.

Q : Puis-je inclure dans mon texte des considérations propres à la langue ou au pays ?

R : Nous préférerions que vous ne le fassiez pas.

Supposons par exemple que vous traduisiez le Manuel de Référence en Coréen et que vous vouliez inclure une section sur les revendeurs en Corée dans votre Manuel.

Il n'y a pas vraiment de raison pour que cette information ne soit pas aussi présente dans la version anglaise (ou allemande, espagnole, ...). Il se peut qu'un anglophone ait besoin d'un exemplaire de FreeBSD alors qu'il se trouve en Corée. Cela permet aussi de mettre en avant la pénétration de FreeBSD dans le monde entier, ce qui n'est pas une mauvaise chose.

Si vous avez des informations propres à un pays donné, soumettez-les d'abord sous forme de modifications à la version anglaise du Manuel de Référence (avec send-pr(1)) et traduisez-les ensuite dans votre langue dans le Manuel de Référence dans cette langue.

Merci.

Q : Comment faut-il coder les caractères propres à une langue ?

Dans les documentations, les caractères Non-ASCII doivent apparaître sous forme d'entités SGML.

Brièvement, celles-ci sont constituées d'une perluette (&), du nom de l'entité, et d'un point-virgule (;).

Les noms des entités sont définis par la norme ISO8879, que vous trouverez dans le catalogue des logiciels portés, sous `textproc/iso8879` .

Voici quelques examples :
Entité: é
L'apparence: é
Définition: “e” accent aigu minuscule
Entité: É
L'apparence: É
Définition: “e” accent aigu majuscule
Entité: ü
L'apparence: ü
Définition: “u” tréma minuscule

Après installation du logiciel porté “iso8879”, le fichier `/usr/local/share/xml/iso8879` en donne la liste complète.

Q : Comment doit-on s'adresser au lecteur ?

R : Dans les documents anglais, le “*you*” est employé, il n'y a qu'une seule forme, à l'inverse d'autres langues.

Si vous traduisez dans une langue qui dispose de plusieurs formes, utilisez celle que l'on emploie habituellement dans les documentations techniques. En cas de doute, servez-vous d'une forme de politesse courante.

Q : Y'a-t-il des informations supplémentaires à inclure dans les traductions ?

R : Oui.

Les en-têtes de la version anglaise du document ressembleront à ceci :

```
<!--
     The FreeBSD Documentation Project

     $Id: chapter.xml,v 1.11 1999/06/20 21:18:57 billf Exp $
-->
```

La forme exacte peut changer, mais elles comporteront toujours la ligne “Id” et la phrase `The FreeBSD Documentation Project` .

Vos traductions doivent comporter leur propre ligne “Id” et `FreeBSD Documentation Project` doit être remplacé par `The FreeBSD Langue Documentation Project`.

Vous devrez aussi ajouter une troisième ligne qui donne la version anglaise d'origine sur laquelle est basée la traduction.

Donc, la version espagnole du présent fichier commencerait par :

```
<--
     The FreeBSD Spanish Documentation Project

     $Id: chapter.xml,v 1.3 1999/06/24 19:12:32 jesusr Exp $
     Original revision: 1.11
-->
```

# Chapitre 10. Style

Pour conserver une certaine cohérence entre le travail des nombreux rédacteurs de la documentation de FreeBSD, on a défini quelques règles à suivre.

N'utilisez pas de formes contractées
: N'utilisez pas de formes contractées. Utilisez toujours les mots entiers. “*Don't use contractions*” n'est pas correct[1].

  Ne pas contracter donne au texte un ton plus formel, est plus précis, et facilite la tâche des traducteurs.

Utilisez la virgule dans les énumérations
: Dans une énumération au sein d'un paragraphe, séparez avec des virgules les éléments les uns des autres. Mettez aussi un virgule et la conjonction “et” avant le dernier élément.

  Examinez par exemple la phrase suivante :

  > C'est une liste d'un, deux et trois éléments.

  Est-ce une liste de trois éléments, “un”, “deux”, et “trois”, ou une liste de deux éléments, “un” et “deux et trois” ?

  Il vaut mieux être explicite et ajouter la dernière virgule :

  > C'est une liste d'un, deux, et trois éléments.

Evitez les redondances
: Evitez les redondances. En particulier, “la commande”, “le fichier”, et “man commande” sont probablement redondants.

  Voici deux exemples pour ce qui concerne les commandes. Il est préférable d'utiliser le second.

  Utilisez la commande `cvsup` pour mettre à jour vos sources.

  Utilisez `cvsup` pour mettre à jour vos sources.

  Voici deux exemples pour ce qui concerne les noms de fichiers. Il est préférable d'utiliser le second.

  … dans le fichier `/etc/rc.local` …

  … dans `/etc/rc.local` …

[1] N.d.T.: Ceci s'applique bien sûr aux textes rédigés en anglais.

Voici enfin deux exemples pour les références aux pages de manuel. Il est préférable d'utiliser le second (il emploie `citerefentry`).

Voyez `man csh` pour plus d'information.

Voyez csh(1)

Pour des détails sur le style, consultez les *Eléments de Style*, de William Strunk.

# Chapitre 11. Utiliser `sgml-mode` avec Emacs

Les versions récentes d'Emacs ou Xemacs (disponibles au catalogue des logiciels portés) incluent un paquetage très utile appelé PSGML. Il est automatiquement appelé au chargement d'un fichier avec l'extension `.xml`, ou lorsque l'on tape `M-x sgml-mode`. C'est un mode majeur pour traiter les fichiers SGML, les éléments et les attributs.

Connaître certaines des commandes de ce mode peut rendre le travail sur des documents comme le Manuel de Référence beaucoup plus facile.

`C-c C-e`
: Exécute `sgml-insert-element`. On vous demandera le nom de l'élement à insérer là ou se trouve le curseur. Vous pouvez utiliser la touche Tab pour compléter le nom de l'élément. Seuls les éléments syntaxiquement valides à cet endroit seront acceptés.

  L'éditeur insérera les marques de début et de fin de l'élément. S'il y a d'autres éléments obligatoires qui doivent être inclus dans cet élément, ils seront aussi inclus.

`C-c =`
: Exécute `sgml-change-element-name`. Mettez-vous dans un élément et utilisez cette commande. On vous demandera le nom de l'élément par lequel il faut le remplacer. Les marques de début et de fin de l'élément seront remplacées.

`C-c C-r`
: Exécute `sgml-tag-region`. Sélectionnez du texte (placez-vous au début, `C-espace`, allez à la fin du texte, `C-espace`) et lancez ensuite cette commande. On vous demandera quel élement utiliser. Celui-ci sera inséré immédiatement avant et après la région choisie.

`C-c -`
: Exécute `sgml-untag-element`. Mettez-vous sur la marque de début ou de fin de l'élément que vous voulez supprimer et lancez cette commande. Les marques de début et de fin de l'élément seront supprimées.

`C-c C-q`
: Exécute `sgml-fill-element`. “Remplira” (i.e., reformatera) le contenu de l'élément courant. Cela affectera aussi le contenu dont les blancs sont significatifs, comme celui des éléments `programlisting`, utilisez donc cette commande avec précaution.

`C-c C-a`
: Exécute `sgml-edit-attributes`. Ouvre un deuxième tampon donnant la liste des attributs de l'élément qui inclut le contenu courant, avec leurs valeurs. La touche Tab vous permet de passer d'un attribut à l'autre, `C-k` de modifier une valeur existante, et `C-c` de fermer le tampon et de revenir au document principal.

`C-c C-v`
: Exécute `sgml-validate`. Vous propose de sauvegarder le document en cours (si besoin est) et passe ensuite un programme de validation du SGML. Les résultats de cette validation sont affichés dans un nouveau tampon et vous pouvez ensuite naviguer d'une erreur à l'autre, pour les corriger au fur et à mesure.

Il y a sans aucun doute d'autres fonctions utiles, mais j'ai décrit celles que j'utilise le plus souvent.

# Chapitre 12. A consulter

Ce document ne décrit délibérement pas exhaustivement, ni le SGML, ni les DTDs citées, ni le Projet de Documentation de FreeBSD. Pour plus d'information sur ces sujets, il est recommandé de consulter les sites Web ci-dessous.

## 12.1. Projet de Documentation de FreeBSD

- Les pages Web du Projet de Documentation de FreeBSD,
- Le Manuel de Référence de FreeBSD.

## 12.2. SGML

- La page Web SGML/XML, un ressource SGML exhaustive,
- L'Introduction en douceur au SGML.

## 12.3. HTML

- Le consortium World Wide Web,
- Les spécifications du HTML 4.0.

## 12.4. DocBook

- Le *Davenport Group*, qui maintient la DTD DocBook.

## 12.5. Le Projet de Documentation de Linux

- Les pages Web du Projet de Documentation de Linux.

# Index