A FreeBSD Dokumentációs Projekt irányelvei kezdőknek

A FreeBSD Dokumentációs Projekt irányelvei kezdőknek

Verzió: 43184
2013-11-13 írta hrs.

Kivonat

Köszönjük a részvételt a FreeBSD Dokumentációs Projektben! Minden segítség nagyon fontos számunkra.

Ebben az ismertetőben megtalálható a FreeBSD Dokumentációs Projekt munkáját segítő (kötelező és ajánlott) szoftverek és segédeszközök leírásától kezdődően a Dokumentációs Projekt mögött álló elképzelések bemutatásáig minden olyan hasznos információ, amelyre szükségünk lehet a munkánk megkezdéséhez.

A leíráson folyamatosan dolgozunk, nem tekinthető még véglegesnek. A befejezetlen szakaszokat a címükben csillaggal jelöltük meg.

Fordította: Páli Gábor, utolsó ellenőrzés: 2010.11.28.

EGYÉB) ERED A DOKUMENTUM AKÁRMIFÉLE FELHASZNÁLÁSÁBÓL, MÉG HA ERRE KÜLÖN FEL IS HÍVTUK a FIGYELMET.

Tartalom

A példák listája

Bevezetés

1. Parancssori promptok

A következő táblázatban láthatjuk a rendszer alapértelmezett promptját és a rendszeradminisztrátor promptját. A példákban ilyen elemek segítségével fogjuk jelezni, hogy milyen felhasználóként kell azokat lefuttatni.

Felhasználó	Prompt
Egyszerű felhasználó	`%`
Rendszeradminisztrátor	`#`

2. Szedési szabályok

Az alábbi táblázatban röviden összefoglaljuk a könyvben alkalmazott szedési irányelveket.

Leírás	Példa
Parancsok	A `ls -l` használatával listázzuk ki az összes állományt.
Állománynevek	Nyissuk meg a `.login` állományt.
Képernyőn megjelenő üzenetek	`You have mail.`
Felhasználói parancsok	`% su` `Password:`
Hivatkozások man oldalakra	A su(1) használatával váltsunk felhasználót.
Felhasználói- és csoportnevek	Ezt kizárólag csak a `root` felhasználó végezheti el.
Kiemelések	Ezt meg *kell* csinálni.
Parancssori változók: helyettesítsük egy valódi névvel vagy változóval	Az állomány törléséhez adjuk ki az `rm állománynév` parancsot.
Környezeti változók	A `$HOME` a saját felhasználói könyvtárunkat tartalmazza.

3. Megjegyzések, tanácsok, fontosabb információk, figyelmeztetések és példák

A szövegben előfordulhatnak megjegyzések, figyelmeztetések és példák.

Megjegyzés

Így jelennek meg a megjegyzések és általában ránk hatással levő információkat tartalmaznak, amelyeket érdemes figyelembe vennünk.

Tipp

Így jelennek meg a gyakorta hasznos tanácsok, amelyek esetenként egy másik, gyakran egyszerűbb megoldást mutatnak be.

Fontos

Így jelennek meg a fontosabb információk. Általában még további elvégzendő lépéseket adnak meg.

Figyelem

Így jelennek meg a figyelmeztetések. Határozottan érdemes rájuk figyelni, mert ha nem követjük pontosan a bennük megadott utasításokat, akkor azzal kárt okozhatunk a rendszerünkben. Ez lehet fizikai, tehát a hardvereszközeink sérülését okozó probléma, vagy nem fizikai, tehát például egy fontos állomány akartlan törlése.

1. példa - Mintapélda

Így jelennek meg a példák, amelyek jellemzően valaminek a részletes bemutatását vagy egy konkrét művelet eredményét tartalmazzák.

4. Köszönetnyilvánítás

Szeretnénk megköszönni Sue Blake, Patrick Durusau, Jon Hamilton, Peter Flynn és Christopher Maden munkáját, akik kellő fordítottak időt arra, hogy átolvassák a könyv kezdeti változatait, majd azt számos értékes megjegyzéssel és javaslattal gazdagítsák.

1. fejezet - Áttekintés

Üdvözöljük a FreeBSD Dokumentációs Projektben! A FreeBSD sikerességéhez elegendhetetlenül fontos a jó minőségű dokumentáció, amelynek jelentős részét a FreeBSD Dokumentációs Projekt (röviden FDP) állítja elő. Ez a projekt rendkívül értékesnek tart bármilyen fajta támogatást.

Ennek a dokumentumnak fő célja, hogy bemutassa *az FDP szerveződését*, *hogyan készítsünk és küldjünk az FDP részére dokumentációt*, és *hogyan használjuk hatékonyan a dokumentációk írásához készített eszközöket*.

Az FDP bárkit szívesen fogad. A tagságnak nincs semmiféle előfeltétele, a tagoknak nem kötelező havonta adott mennyiségű dokumentációk írniuk. A belépésnek mindössze annyit kell tennünk, hogy feliratkozunk a FreeBSD Dokumentációs Projekt levelezési lista tagjai közé.

A könyv elolvasása során megismerjük:

- milyen dokumentációkat tart karban az FDP;
- az FDP által karbantartott dokumentációk megírásához használt SGML nyelvet;
- hogyan készítsünk módosításokat a dokumentációhoz;
- hogyan küldjük be és nézessük át a módosításainkat, ezek miként kerülhetnek be a FreeBSD hivatalos dokumentációjába.

1.1. A FreeBSD dokumentációja

Az FDP a FreeBSD-hez mellékelt dokumentációk négy fajtájáért felelős:

Man oldalak
: Az angol nyelvű man oldalakat nem az FDP tagjai készítik, azok az alaprendszer részei. Az FDP viszont az olvashatóság vagy az esetleges pontatlanságok javítása érdekében át tudja fogalmazni ezeket.

  Továbbá a különböző fordítói csoportok felelősek a rendszerhez tartozó egyes man oldalak fordításaiért, amelyek szintén az FDP keretein belül készülnek.

GYIK
: A GYIK célja a FreeBSD-vel foglalkozó különböző levelező listákon és hírcsoportokon felbukkanó gyakran ismételt kérdések (rövid, lényegretörő) megválaszolása. Ez a formátum nem teszi lehetővé hosszabb és minden részletre kiterjedő válaszok megfogalmazását.

Kézikönyv
: A kézikönyv a FreeBSD felhasználók számára összeállított átfogó, intereten keresztül is olvasható referencia.

Honlap
: A http://www.FreeBSD.org/ címen, illetve annak különböző tükrözésein keresztül elérhető honlapok képviselik a FreeBSD elsődleges megjelenési formáját a világhálón. Legtöbben ezen az oldalon találkoznak a FreeBSD-vel először.

Az imént felsorolt kategóriák mindegyike megtalálható a FreeBSD repositoryjában, ezért a velük kapcsolatos változtatásokhoz tartozó naplóbejegyzések mindenki számára láthatóak, illetve a CVSup vagy CTM alkalmazások használatával a dokumentációkból létre tudunk hozni egy helyi másolatot.

Mindezek mellett sokan készítettek további oktatóanyagokat és FreeBSD-hez kapcsolódó honlapokat. Ezek némelyike (a szerző előzetes engedélyével) a repositoryban is megtalálható. A többi esetben a szerzők úgy döntöttek, hogy a FreeBSD forrásaitól független helyen szeretnék tárolni a leírásaikat. Ilyenkor pedig az FDP törekszik belinkelni ezeket.

1.2. Mielőtt belekezdenénk

Az alábbi ismereteket feltételezzük az olvasó részéről:

- Képesek vagyunk lekérni és folyamatosan frissíteni a FreeBSD repository egy helyi számítógépen tárolt változatát (például a CVS, CVSup vagy CTM alkalmazások valamelyikének használatával), vagy egyszerűen csak letölteni belőle a CVSup segítségével egy *előre kikért* példányt.

- Tudunk szoftvereket telepíteni a FreeBSD Portgyűjteményéből vagy a pkg_add(1) használatával.

1.3. A legfontosabb tudnivalók

Ha most csak a legalapvetőbb ismeretekre van szükségünk, és a részletekkel csupán később akarunk majd foglalkozni, akkor az alábbi utasítások alapján javasolt elindulni:

1. Telepítsük a textproc/docproj metaportot:

```
# cd -/usr/ports/textproc/docproj
# make JADETEX=no install
```

2. Készítsünk egy helyi másolatot a FreeBSD forrásának `doc` könyvtáráról. Erre megfelel a CVSup `checkout` módja vagy a repository teljes tartalmának letöltése.

Egy használható helyi másolathoz legalább a `doc/share` és `doc/en_US.ISO8859-1/share` könyvtárakat kell kikérnünk:

```
% cvs checkout doc/share
% cvs checkout doc/en_US.ISO8859-1/share
```

Ha viszont sok tárhellyel rendelkezük, akkor akár mindent kikérhetünk:

```
% cvs checkout doc
```

3. Amikor egy korábban felrakott könyvhöz vagy cikkhez készítünk módosításokat, előtte kérjük ki a repositoryból. Ha egy új könyvet vagy cikket szeretnék beküldeni, akkor építkezzünk a meglevőekből.

   Tegyük fel például, hogy írtunk egy cikket a FreeBSD és Windows® 2000 rendszerek közti VPN-hálózatok létrehozásáról:

   1. Kérjük ki az `articles` könyvtár tartalmát.

      ```
      % cvs checkout doc/en_US.ISO8859-1/articles
      ```

   2. Másoljunk le valamelyik cikket és használjuk fel sablonként. Most úgy döntöttük, hogy az új cikket a `vpn-w2k` könyvtárba helyezzük el:

      ```
      % cd doc/en_US.ISO8859-1/articles
      % cp --R committers-guide vpn-w2k
      ```

   Ha viszont egy már létező dokumentumot akarunk szerkeszteni, például a GYIK-ot, akkor kérjük ki azt a könyvtárat, amelyikben található. Ez ebben az esetben a `doc/en_US.ISO8859-1/books/faq` :

   ```
   % cvs checkout doc/en_US.ISO8859-1/books/faq
   ```

4. A könyvtárban található `.xml` állományokat írjuk át a kedvenc szövegszerkesztőnkkel.

5. A forrásban használt jelölőket a `lint` cél segítségével ellenőrizhetjük. Ilyenkor anélkül tudjuk megtalálni a dokumentum forrásában rejtőző hibákat, hogy ténylegesen el kellene végeznünk azok időigényes átalakítását:

   ```
   % make lint
   ```

   Amikor viszont valóban le akarjuk generáltatni a dokumentumot, a `FORMATS` változóban tudjuk felsorolt a kért formátumokat. Itt jelen pillanatban a `html`, `html-split`, `txt`, `ps`, `pdf` és `rtf` értékeket szerepeltethetjük. A támogatott formátumok legfrissebb listáját mellesleg a `doc/share/doc.docbook.mk` állományban találhatjuk meg. Ha egyszerre több formátumot kértünk, akkor ne felejtsük el idézőjelek közé tenni a felsorolást.

Például így lehet egy dokumentumot csak `html` formátumra alakítani:

```
% make FORMATS=html
```

Ha viszont egyszerre szeretnénk a dokumentumból `html` és `txt` formátumot létrehozatni, akkor megtehetjük azt két külön make(1) paranccsal:

```
% make FORMATS=html
% make FORMATS=txt
```

Vagy egyetlen paranccsal:

```
% make FORMATS="html txt"
```

6. Küldjük be a módosításainkat a send-pr(1) használatával.

2. fejezet - Eszközök

Az FDP a FreeBSD dokumentációját többféle eszköz segítségével tartja karban, alakítja át különböző kimeneti formátumokra és így tovább. Ha tehát a FreeBSD dokumentációjával akarunk dolgozni, akkor mindezekre az eszközökre nekünk is szükségünk lesz.

Ezek az eszközök a FreeBSD csomag- és portgyűjteményében is megtalálhatóak, ami így nagyban megkönnyíti a telepítésüket.

Tehát érdemes elvégeznünk a telepítésüket mielőtt foglalkoznánk a későbbi fejezetekben található példákkal. Az egyes programok konkrét használatával is majd ezekben a fejezetekben fogunk részletesebben foglalkozni.

Lehetőség szerint a textproc/docproj portot használjuk

A textproc/docproj port telepítésével rengeteg időt és fáradalmat megtakaríthatunk magunknak. Ez egy ún. *metaport*, amely önmaga nem tartalmaz semmilyen szoftvert, helyette azonban függ az egyébként telepítendő portoktól. Így tehát csupán ezen port telepítésével automatikusan le *kellene* töltődnie és települnie *kellene* a fejezetben ismertetett összes csomagnak.

Az egyik telepítésre javasolt csomag a JadeTeX elnevezésű makrókészlet, amelynek viszont szüksége van a TeX csomagra. A TeX egy viszonylag nagy méretű csomag, ennek a tényleges telepítését csak abban az esetben javasoljuk, ha a dokumentációból Postscript vagy PDF változatot akarunk készíteni.

Telepítési idő és tárterület szempontjából nyilatkoznunk kell róla, hogy a port részeként a JadeTeX (és így a TeX) felkerüljön vagy sem. Ennek megfelelően választhatunk:

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

vagy

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

Ugyanezt a választást a textproc/docproj-jadetex vagy a textproc/docproj-nojadetex portok valamelyikének telepítésével is megtehetjük. Ezek a segédportok helyettünk már definiálják a `JADETEX` változó értékét, és ennek megfelelően telepítik gépünkre az alkalmazásokat. Ne felejtsük el, hogy ha nem tesszük fel a JadeTeX csomagot, akkor csak HTML és ASCII formátumú

dokumentációt leszünk képesek előállítani. Postscript vagy PDF készítéséhez mindenképpen szükséges a TeX.

2.1. Alapeszközök

2.1.1. Szoftverek

A FreeBSD dokumentációjával csak az ebben a szakaszban ismertetett programok segítségével tudunk érdemben dolgozni. Ezekkel a programokkal tudjuk lényegében átalakítani a dokumentációt többek közt egyszerű ASCII szöveggé, HTML oldalakká vagy RTF dokumentumokká. Mindegyikük része a textproc/docproj csomagnak.

Jade (textproc/jade)
: Egy DSSSL implementáció, ezen keresztül alakíthatóak át a dokumentumok jelölőkkel ellátott forrásai más, például HTML vagy TeX formátumokba.

Tidy (www/tidy)
: Egy HTML forrásokra alkalmazható „formázó”, amellyel a többi program által automatikusan létrehozott egyes HTML állományokat lehet emberek számára könnyebben érthető alakra hozni.

Links (www/links)
: Egy szöveges módban működő webböngésző, amely remekül használható HTML oldalak egyszerű szöveges változatainak létrehozására.

peps (graphics/peps)
: A dokumentációban néhol találhatóak ábrák, amelyek némelyike EPS állományokban tárolódik. A webböngészők azonban csak akkor fogják tudni ezeket megjeleníteni, ha előtte átalakítjuk PNG állományokká.

2.1.2. Dokumentumtípus-definíciók és egyedek

Az FDP az itt felsorolt dokumentumtípus-definíciókat (DTD-ket) használja. A dokumentációval csak ezek telepítése után tudunk dolgozni.

HTML DTD (textproc/html)
: A HTML a World Wide Web nyelveként egységesen elfogadott jelölőnyelv, amely ezáltal a FreeBSD honlapjának is alapja.

DocBook DTD (textproc/docbook)
: A DocBook a különféle szakmai jellegű dokumentációk készítéséhez kialakított jelölőnyelv. A FreeBSD teljes dokumentációja DocBook formátumban készül.

ISO 8879 szabványú egyedek (textproc/iso8879)
: Az ISO 8879:1986 szabványban meghatározott karakteregyed-készletek közül 19 előfordul számos DTD részeként. Ezekben szerepelnek matematikai szimbólumok, a Latin karakterkészletekben megjelenő további (ékezetes, mellékjeles stb.) karakterek és görög szimbólumok.

2.1.3. Stíluslapok

A stíluslapok felhasználásával tudjuk a képernyő, a nyomtatás stb. számára alkalmassá tenni a dokumentációkat az átalakítás vagy a formázás során.

Moduláris DocBook stíluslapok (textproc/dsssl-docbook-modular)
: A moduláris DocBook stíluslapok alkalmazásával alakítjuk át a DocBook formában előkészített dokumentációt más, például HTML vagy RTF változatúra.

2.2. Kiegészítő eszközök

Ebben a szakaszban további választható eszközöket sorolunk fel. Telepítésük nem kötelező, azonban jelentős mértékben meg tudják könnyíteni a munkánkat, illetve a dokumentációból előállítható kimeneti formátumok terén kínálnak nagyobb rugalmasságot.

2.2.1. Szoftverek

JadeTeX és teTeX (print/jadetex és print/teTeX)
: A Jade és a teTeX alkalmazások segítségével alakíthatóak át a DocBook dokumentumaink DVI, Postscript és PDF állományokká. Ehhez viszont a JadeTeX makrókat is telepítenünk kell.

  Ha az imént említett formátumok egyikére sincs szükségünk (tehát elegendő a HTML, szöveges és RTF kimenet), akkor nem kell telepítenünk a JadeTeX és teTeX szoftvereket. Ezzel egyébként viszonylag sok helyet megspórolhatunk, mivel a teTeX közel 30 MB méretű.

Fontos

Ha a JadeTeX és a teTeX telepítése mellett döntünk, akkor a JadeTeX telepítése után megfelelően be kell állítanunk a teTeX alkalmazást. Az erre vonatkozó részletes utasításokat a `print/jadetex/pkg-message` állományban olvashatjuk.

Emacs vagy XEmacs (editors/emacs vagy editors/xemacs)
: Mind a két szövegszerkesztő tartalmaz az SGML DTD-hez igazodó speciális szerkesztési módot. Ebben olyan parancsok találhatóak, amelyekkel csökkenthető a munka elvégzéséhez szükséges gépelés és ezáltal a hibák keletkezésének valószínűsége.

  Egyáltalán nem kötelező ezeket használni. A feladatra bármilyen szabadon választott szövegszerkesztő tökéletesen megfelelő, viszont a fentiek némileg megkönnyíthetik a munkavégzést.

Ha ismerünk az SGML dokumentumok feldolgozása során alkalmazható további hasznos szoftvereket, akkor jelezzük bátran a Documentation Engineering Team <doceng@FreeBSD.org > felé és felveszik erre a listára.

3. fejezet - SGML alapismeretek

Az FDP keretében készített dokumentációk többsége az SGML valamilyen alkalmazásában íródik. Ebben a fejezetben részletesebben kifejtjük a mögötte álló fogalmakat, a dokumentumok alapjául szolgáló források megértését és írását, illetve a dokumentáció forrásainak tanulmányozása során előkerülő különféle SGML-trükköket.

A bemutatás alapjául szolgáltak Mark Galassi Get Going With DocBook című írásának egyes részei.

3.1. Áttekintés

A kezdeti időkben még viszonylag könnyen el lehetett boldogulni az elektronikus formában tárolt szövegekkel. Elegendő volt csupán annyit tudni., hogy az adott írást milyen karakterkódolással készítették (ez lehetett ASCII, EBCDIC vagy éppen valami más). A szöveg nem volt több mint egyszerű szöveg, és közvetlenül a végleges formáját adta. Semmi csel, semmi formázás, semmi hozzáadott értelem.

Ezen a fokon aztán elkerülhetetlen módon tovább kellett lépni. Hiszen ha egyszer a szöveges információkat egy számítógép által kezelhető alakban tároljuk, akkor jogosan elvárhatjuk, hogy az képes legyen felhasználni és értelmesen feldolgozni. Szeretnénk a szöveg bizonyos részeit például kiemelni, felvenni egy szójegyzékbe, vagy éppen hivatkozással ellátni. Az állományok neveit a képernyőn „írógépszerű”, a nyomtatásban viszont már „dőltbetűs” stílusban szeretnénk látni, nem is beszélve a szöveg megjelenésének számtalan egyéb módjáról.

Egy időben a Mesterséges Intelligencia (MI) megjelenésétől várták a megváltást ezen a területen. A számítógépünk majd szépen beolvassa az általunk írt dokumentumot és magától felismeri a fontosabb kulcsszavakat, állományneveket, a felhasználó által begépelendő szövegeket, a példákat és így tovább. Sajnálatosan azonban a valóságban ez még egyáltalán nem valósult meg, a számítógépeknek ezért szükségünk van némi segítségre a szöveges adatok értelmes feldolgozásában.

Pontosabban úgy fogalmazhatnánk, hogy segítenünk kell nekik az egyes elemek beazonosításában. Nézzük meg például ezt a szöveget:

> Az rm(1) parancs használatával töröljük a `/tmp/ize` állományt:
>
> ```
> % rm -/tmp/ize
> ```

Emberi szemmel könnyedén fel tudjuk ismerni benne az állományneveket, a parancsokat, a man oldalak hivatkozásait és így tovább, azonban a számítógép erre önállóan nem képes. Ezért lesz szükségünk jelölőkre.

A „jelölő” szó eredetijét (markup) gyakran olyan értelemben használják mint „haszonkulcs” vagy „kockázati pótlék”. Kevés elvonatkoztatással ugyanez lényegében alkalmazható a szövegek esetében is. A jelölők a dokumentumban szereplő kiegészítő, hasznos, az azonosítás kockázatát csökkentő, a szöveg többi részétől egyértelműen megkülönböztethető további szöveges információkat jelentik. Ezek alapján a programok a dokumentumok feldolgozása során képesek önállóan meghozni bizonyos döntéseket. A szövegszerkesztők el tudják rejteni ezeket a többletinformációkat az olvasók elől, így azok egyáltalán nem zavarják őket.

A jelölőkben tárolt adatok tehát *növelik a dokumentumok hasznát*. A jelölők hozzáadását, a szöveg bejelölését értelemszerűen emberek végzik, hiszen ha erre a számítógépek is képesek lennének, akkor nem is lenne rájuk egyáltalán szükség. Ezzel azonban *pótlékot kell nyújtanunk* (vagyis további költségeket ráfordítanunk) a dokumentumok megírásához.

Az előző példában szereplő szöveget ennek megfelelően a következő módon írjuk meg:

```
<para>Az &man.rm.1; parancs használatával
  töröljük a <filename>/tmp/ize</filename>
  állományt:</para>

<screen>&prompt.user; <userinput>rm -/tmp/ize</userinput></screen>
```

Láthatjuk, hogy a jelölők nagyon jól elkülöníthetőek a szöveg tartalmától.

A jelölők használatához nyilvánvalóan valamilyen módon meg kell határoznunk, hogy az adott jelölők mit jelentenek és hogyan kell azokat értelmezni. A jelölők összefogásához tehát szükségünk van egy ún. jelölőnyelvre, amely alapján aztán jelölni fogjuk a dokumentumainkat.

Ehhez természetesen egyetlen jelölőnyelv önmagában még nem feltétlenül lesz elég. A szaknyelven íródott dokumentációkhoz igazított jelölőnyelvvel szemben teljesen másak az elvárásaink, mint például a receptek leírásához használt nyelv esetében, ez pedig megint más, mint amivel verseket tudunk jelölni. Először tehát egy olyan nyelvet kell megfogalmaznunk, amely ilyen jelölőnyelvek előírására használható. Ezt nevezzük a jelölőnyelvek jelölőnyelvének, vagyis a *meta-jelölőnyelvnek*.

Az SGML, avagy *Standard Generalized Markup Language* (Szabványos Általánosított Jelölőnyelv) pontosan egy ilyen nyelv. Számos jelölőnyelv készült az SGML segítségével, többek közt az FDP által leginkább használt HTML és DocBook.

Az egyes nyelvek részletes leírását hivatalosan dokumetumtípus-definíciónak (*Documentum Type Definition*, *DTD*) nevezik. A DTD felhasználásával adhatjuk meg a szövegben jelölőként alkalmazható elemeket, azok sorrendjét (vagy éppen egymásba

ágyazhatóságának mikéntjét) és a hozzájuk kapcsolódó egyéb információkat. A DTD-ket gyakran csak úgy említik mint az SGML *alkalmazásait*.

A DTD tartalmazza az *összes* felhasználható elem leírását, azok használatának sorrendjét, megadja, hogy ezek közül melyeknek kell szerepelniük, illetve melyek hagyhatóak el és így tovább. Ennek köszönhetően készíthető egy olyan SGML alapján működő *elemző*, amely a DTD és egy dokumentum birtokában képes megállapítani, hogy az adott dokumentum megfelel-e a DTD által meghatározott szabályoknak: a benne szereplő elemek a megfelelő sorrendben vannak, esetleg tartalmaznak hibákat. Ezt a lépést nevezik általában a „dokumentum érvényesítésének".

Megjegyzés

Az ellenőrzés folyamán egyszerűen annyi történik, hogy az elemző a megadott DTD alapján jóváhagyja a dokumentumban feltüntetett elemeket, azok rendezettségét és a többit. A jelölők *helyes* használatát azonban *nem* vizsgálja. Ha éppen függvénynévként jelöljük be a szövegben megjelenő állományok neveit, akkor az elemező ezt nem fogja hibának tekinteni (ekkor természetesen feltételezzük, hogy a DTD definiálja az állomány- és függvénynevek jelölésére alkalmas elemeket, illetve ezek ugyanazokon a helyeken szerepelhetnek).

A Dokumentációs Projekt számára beküldött munkáinkban jó eséllyel a HTML vagy a DocBook nyelvek valamelyike szerint kell dokumentumokat megjelölnünk, és nem kell a DTD módosításával foglalkoznunk. Ennélfogva ez a leírás sem tér ki a DTD írásának részleteire.

3.2. Elemek, címkék és tulajdonságok

Az SGML használatával készített dokumentumtípus-definíciók mindegyikének vannak közös jellemzői. Ez viszont aligha lesz számunkra meglepő, ahogy majd fokozatosan megismerkedünk az SGML kialakítása mögött álló alapvető gondolatokkal. Ezek közül a legkézenfekvőbbek a *tartalom* és az *elem*.

A dokumentáció minden esetben (legyen az most egy normál honlap vagy éppen egy vaskos könyv) rendelkezik valamilyen tartalommal, amelyet aztán tovább (esetleg még tovább) osztunk elemekre. A jelölők elhelyezésének ezen elemek határainak kijelölésében és elnevezésében van szerepe a feldolgozás későbbi szakaszaiban.

Ehhez példaként tekintsünk egy hagyományos könyvet. A legfelső szinten ez a könyv önmagában egy elemet képvisel. Ez a „könyv" elem aztán magától értetődő

módon tartalmaz fejezeteket, amelyek szintén önálló elemeknek tekinthetőek. Minden ilyen fejezet további elemeket foglal magában, például bekezdéseket, idézeteket és lábjegyzeteket. Minden egyes bekezdésben találhatunk újabb elemeket, amelyek elárulják nekünk, hogy a bennük szereplő szövegben melyik részében beszélnek egymással a szereplők, vagy éppen hogy hívják az egyes karaktereket.

Az egészet úgy képzelhetjük el mint a tartalom „feldarabolását". A legfelső szinten adott egy darab, maga a könyv. Ahogy haladunk kicsivel lentebb, újabb darabokat találunk, a fejezeteket. Ezeket aztán tovább bomlanak bekezdésekre, lábjegyzetekre, a karakterek neveire és a többi.

Meglepő, hogy az SGML lehetőségeinek igénybevétele nélkül milyen könnyen különbséget tudunk tenni az egyes elemek közt. Ehhez valójában elegendő a könyv nyomtatott változata, néhány különböző színű kiemelő, amelyekkel aztán bejelöljük a tartalom egyes részeit.

Sajnos a kiemelőknek nem létezik elektronikus változata, ezért találnunk kell valamilyen egyéb módot a tartalom egyes részeinek megjelölésére. Az SGML-ben megfogalmazott nyelvek (HTML, DocBook és társaik) ezt *címkékkel* oldják meg.

A címkékkel mondhatjuk meg hol kezdődnek és hol fejeződnek be az egyes elemek. *A címke nem az elem része.* Mivel a DTD általában azért készül, hogy a szövegben adott típusú információkat tudjunk jelölni, adott típusú elemeket fog elfogadni, ezért ezeknek megfelelően kell címkéket létrehoznunk.

Egy *`elem`* elemhez tartozó kezdőcímke általános alakja az *`elem`*. Az hozzá tartozó zárócímke pedig az */`elem`*.

3.1. példa - Elem (kezdő- és zárócímkék) használata

A HTML-ben a bekezdéseket a `p` (mint paragrafus) elemmel jelölhetjük. Ehhez az elemhez tartozik kezdő- és zárócímke.

```
<p>Ez egy bekezdés.  A -'p' elem kezdőcímkéjétől indul és ↺
a -'p'
  zárócímkéjénél fejeződik be.</p>

<p>Ez meg egy másik bekezdés.  Ez viszont már rövidebb.</p>
```

Nem mindegyik elemnél kell zárócímkét használnunk, egyes elemekhez ugyanis nem járul semmilyen tartalom. Például egy HTML állományban jelölhetjük, hogy legyen a dokumentumban egy vízszintes elválasztó. Ehhez a vonalhoz értelemszerűen nem kapcsolódik tartalom, ezért elég egy kezdőcímkét beszúrni.

3.2. példa - Elem (csak kezdőcímke) használata

A HTML-ben van egy `hr` nevű elem, amellyel vízszintes elválasztókat (horizontal rule) jelölhetünk. Ennek az elemnek nincs tartalma, ezért csak kezdőcímkével rendelkezik.

```
<p>Ez itt egy bekezdés.</p>

<hr>

<p>Ez pedig egy másik bekezdés.  Az előző bekezdéstől egy ↺
vízszintes
  vonal választja el.</p>
```

Ha eddig még nem sejtettük volna, megemlítjük, hogy az elemek természetesen elemeket is tartalmazhatnak. A korábbi könyves példánkban a könyv elem magában foglalta az összes fejezet elemet, amelyek pedig a bekezdés elemeket és így tovább.

3.3. példa - Elemek elemekben, az `em` elem

```
<p>Ez egy egyszerű <em>bekezdés</em>, amelyben néhány ↺
<em>szót</em>
  szépen <em>kiemeltünk</em>.</p>
```

A DTD pontosan tartalmazza mely elemek tartalmazhatnak további elemeket, valamint az elemek egymásba ágyazhatóságának szabályait.

Fontos

Az emberek gyakran összetévesztik a címkéket az általuk jelölt elemekkel, és egymás szinonímájaként használják ezeket a kifejezéseket. Ez viszont helytelen.

A dokumentumokat elemekből építjük fel. Minden elem előre meghatározott módon kezdődik és fejeződik be. Az elemek kezdetét és végét címkék jelölik.

> Amikor ez a dokumentum (vagy bárki, az SGML használatában járatos személy) „a `p` címkére” hivatkozik, akkor ez alatt a `<`, `p`, `>` karakterekből álló sorozatot érti. Ezzel szemben viszont „a `p`” a teljes elemre vonatkozik.
>
> Ez egy *nagyon* kicsi eltérés, de mindig tartsuk észben!

Az elemeknek lehetnek tulajdonságaik. A tulajdonságokat nevek és értékek párosai alkotják, segítségükkel az elemhez fejthetünk ki további információkat. Ez lehet az adott elem által jelölt tartalom megjelenítésére vonatkozó utasítás, esetleg az elem valamilyen azonosítója vagy valami más.

Az elemek tulajdonságait mindig az adott elem kezdőcímkéjén *belül* soroljuk fel, `tulajdonság="érték"` alakban.

A HTML újabb változataiban például a `p` elemnek van egy `align` tulajdonsága, amely a HTML megjelenítése során javasolja, hogy az általa jelölt bekezdést merre igazítsuk.

Ez az `align` tulajdonság négy előre meghatározott érték valamelyikét kaphatja meg: `left` (balra zárt), `center` (középre zárt), `right` (jobbra zárt) és `justify` (sorkizárt). Ha nem adjuk meg a tulajdonság értékét a kezdőcímkében, akkor alapértelmezés szerint `left` lesz.

3.4. példa - Tulajdonság használata elemben

```
<p align="left">Az -'align' tulajdonság ebben a bekezdésben ↺
igazából
  teljesen felesleges, hiszen alapértelmezés szerint is balra ↺
zárt
  lenne.</p>

<p align="center">Ennek viszont már középre kellene kerülnie.</
p>
```

Egyes tulajdonságok csak adott értékeket vehetnek fel, mint például `left` vagy `justify` , másoknál viszont lényegében bármit megadhatunk. Ha a tulajdonság értékének megfogalmazása során idézőjeleket (`"`) is használni akarunk, akkor az egész kifejezést tegyük egyszeres idézőjelbe.

3.5. példa - A tulajdonságok értékének megadása egyszeres idézőjellel

```
<p align='right'>Jobbra zárt!</p>
```

Előfordulhat, hogy az érték megadásakor egyáltalán nem kell semmilyen idézőjelet használni. Ennek szabályai viszont nagyon halványak, ezért sokkal egyszerűbb *mindig* idézőjelbe tenni a tulajdonságok értékeit.

Az elemekhez, címkékhez és tulajdonságokhoz tartozó információk SGML katalógusokban kerülnek tárolásra. A Dokumentációs Projektben használt eszközök ilyen katalógusok mentén nézik át a munkánkat. A textproc/docproj csomagban a segédprogramok mellett rengeteg ilyen SGML-katalógust találhatunk. A FreeBSD Dokumentációs Projektnek is vannak saját katalógusai. Az alkalmazott eszközöknek mind a két fajta katalógusokat ismerniük kell.

3.2.1. Egy kis gyakorlás...

A szakaszban szereplő példák kipróbálásához telepítenünk kell bizonyos szoftvereket, illetve beállítani egy környezeti változó értékét.

1. Töltsük le és telepítsük a textproc/docproj portot a FreeBSD Portgyűjteményéből. Ez portoknak a portja, tehát egy *metaport*, így a Dokumentációs Projektben használt összes eszköz rajta keresztül letöltődik és telepítődik.

2. A parancssorunk konfigurációs állományában állítsuk be az `SGML_CATALOG_FILES` környezeti változó értékét. (Amennyiben nem az angol nyelvű dokumentációval dolgozunk, itt érdemes a nyelvünknek megfelelő könyvtárakat megadni.)

3.6. példa - Minta **`.profile`** állomány sh(1) és bash(1) parancssorokhoz

```
SGML_ROOT=/usr/local/share/xml
SGML_CATALOG_FILES=${SGML_ROOT}/jade/catalog
SGML_CATALOG_FILES=${SGML_ROOT}/docbook/4.1/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
```

```
SGML_CATALOG_FILES=/usr/doc/share/xml/catalog:
$SGML_CATALOG_FILES
SGML_CATALOG_FILES=/usr/doc/en_US.ISO8859-1/share/xml/
catalog:$SGML_CATALOG_FILES
export SGML_CATALOG_FILES
```

3.7. példa - Minta **`.cshrc`** állomány csh(1) és tcsh(1) parancssorokhoz

```
setenv SGML_ROOT -/usr/local/share/xml
setenv SGML_CATALOG_FILES ${SGML_ROOT}/jade/catalog
setenv SGML_CATALOG_FILES ${SGML_ROOT}/docbook/4.1/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/html/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES ${SGML_ROOT}/iso8879/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES -/usr/doc/share/xml/catalog:
$SGML_CATALOG_FILES
setenv SGML_CATALOG_FILES -/usr/doc/en_US.ISO8859-1/share/
xml/catalog:$SGML_CATALOG_FILES
```

A módosítások elvégzése után vagy jelentkezzük ki majd be, vagy pedig adjuk ki a közvetlenül parancssorban az adott parancsokat.

1. Hozzunk létre egy `próba.xml` nevű állományt, és írjuk bele az alábbi szöveget:

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

<html>
  <head>
    <title>Próba HTML állomány</title>
  </head>

  <body>
    <p>Ebben a bekezdésben legyen valamennyi szöveg.</p>

    <p>Az utána következő bekezdésbe is rakjunk még valamennyi ↺
szöveget.</p>

    <p align="right">Ennek a bekezdésnek jobbra zártnak kellene ↺
lennie.</p>
  </body>
</html>
```

2. Próbáljuk meg az állományt érvényesíteni valamelyik SGML elemezővel.

   A textproc/docproj csomagnak része az `onsgmls` nevű érvényesítést végző elemező [11]. Az `onsgmls` beolvas egy tetszőleges SGML DTD szerint definiált elemekkel jelölt dokumentumot és ebből elkészíti a hozzá tartozó elemstruktúra-információs halmazt (Element Structure Information Set, ESIS, de ezzel itt most nem foglalkozunk).

   Ha viszont az `onsgmls` parancsnak megadjuk a `-s` paramétert, akkor nem generál tényleges eredményt, csupán a hibaüzenetek jeleníti meg. Ennek köszönhetően könnyen ellenőrizni tudjuk, hogy az általunk készített dokumentum érvényes vagy sem.

   Az `onsgmls` parancs használatával tehát ellenőrizzük az imént létrehozott dokumentumunk érvényességét:

   ```
   % onsgmls --s próba.xml
   ```

   Láthatjuk, hogy az `onsgmls` nem jelez semmiféle hibát, ami azt jelenti, hogy a dokumentumunk valóban érvényes.

3. Nézzük meg mi történik akkor, ha kihagyjuk a kötelező elemeket. Töröljük például a `title` és `/title` címkéket, majd próbáljuk meg újra az érvényesítést.

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

   Az `onsgmls` által generált hibaüzenetek kettőspontokkal tagolt csoportokba vagy oszlopokba sorolhatóak.

Oszlop	Jelentés
1	A hibát jelző program neve. Ez minden esetben az `onsgmls`.
2	A hibát tartalmazó állomány neve.
3	A hibát tartalmazó sor száma.
4	A hibát tartalmazó oszlop száma.
5	A generált üzenet jellegét megadó egybetűs kód. Az `I` információt, a `W` figyelmeztetést, az `E` hibát [a], végül pedig az `X` a kereszthivatkozást jelez. Ebből megállapítható, hogy az iménti üzenetek hibákra vonatkoznak.

Oszlop	Jelentés
6	Az üzenet szövege.

[a]Ez nem minden esetben az ötödik oszlopban szerepel. Az `onsgmls -sv` például az `onsgmls:I: "OpenSP" version "1.5.2"` üzenetet adja vissza (a tényleges verziójától függően). Ez például egy információs üzenet.

Egyedül a `title` címke elhagyásával két különböző hibát kaptunk.

Ezek közül az első jelzi, hogy az SGML elemző olyan helyen találkozott tartalommal (amely ebben esetben konkrétan karaktereket jelent és nem az elemet bevezető kezdőcímkét), ahol valami másra számított. Az elemző itt ugyanis valamelyik, a `head` elemen belül szabályosan elhelyezhető elem kezdőcímkéjét várja (amilyen például a `title`).

A második hibát pedig azért kaptuk, mert a `head` elemeknek tartalmazniuk *kell* `title` elemet. Az `onsgmls` ezt azonban nem ebben a formában közli: mivel az elemet még a befejeződése (tehát a `title` megemlítése) előtt lezártuk, szerinte egyszerűen csak nem ért véget rendesen.

4. Tegyük vissza a `title` elemet.

3.3. A DOCTYPE deklarációk

A dokumentumok elején mindig meg kell adni annak a dokumentípus-deklarációnak a nevét, amely alapján készítjük. Ennek köszönhetően az SGML elemzők elő tudják keresni a dokumentum érvényesítéséhez kellő DTD-t.

Ezt az információt általában egyetlen sorban, a DOCTYPE deklarációban adjuk meg.

A HTML DTD 4.0 változatának megfelelő dokumentumokat tehát például így vezetjük be:

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

Ebben a sorban több különböző típusú alkotórészt fedezhetünk fel.

`<!`
: Ez egy *jelzés*, amellyel jelezzük egy SGML-beli deklaráció kezdetét. Ez a sor a dokumentum típusát határozza meg.

`DOCTYPE`
: A dokumentumtípus SGML-beli deklarációját vezeti be.

`html`
: A dokumentumban elsőként megjelenő elemet nevezi meg.

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

Megadja a dokumentum DTD-jéhez tartozó formális publikus azonosítót (*Formal Public Identifier*, *FPI*). Az SGML elemző ennek alapján találja meg a dokumentum feldolgozása során szükséges DTD-t.

A `PUBLIC` nem része az azonosítónak, azonban segít az SGML elemzőnek megtalálni a benne megfogalmazott DTD-t. Később további módszereket is láthatunk majd erre.

`>`

A deklaráció lezárása.

3.3.1. Formális publikus azonosítók

Megjegyzés

Ebben a szakaszban csupán kiegészítő ismeretek szerepelnek. Ezek viszont hasznos hátteret adhatnak például olyan esetekben, amikor ki akarjuk deríteni, hogy az SGML feldolgozó miért nem éri el a megadott DTD állományokat.

A formális publikus azonosítók (FPI) egy speciális felírással rendelkeznek, amely a következő:

```
"Tulajdonos //Kulcsszó  Leírás //Nyelv"
```

`Tulajdonos`

Ez határozza meg kihez tartozik az FPI.

Ha az értéke az „ISO” részlettel kezdődik, akkor az FPI az ISO tulajdona. Például a `"ISO 8879:1986//ENTITIES Greek Symbols//EN"` esetén a görög szimbólumokat tartalmazó egyedkészlet tulajdonosa. Az ISO 8879:1986 az SGML szabvány ISO száma.

Minden más esetben az értéke a következő alakú lesz: `-//Tulajdonos` vagy `+//Tulajdonos` (vegyük észre, hogy a két változat csak a sor elején levő `+` és `-` jelekben tér el).

Ha az érték a `-` jellel kezdődik, akkor a tulajdonos adatai nem regisztráltak, a `+` esetében pedig regisztráltak.

Az ISO 9070:1991 szabvány definiálja a regisztrált nevek előállítását, amelynek értelmében egy ISO publikáció, egy ISBN kód vagy az adott szervezet ISO 6523 szabványnak megfelelő kódjának számából kell származtatni, illetve a nevek kiosztását egy erre felhatalmazott szerv végzi. Ezt a feladatot az Amerikai Nemzeti Szabványügyi Intézetre (ANSI) bízta az ISO tanácsa.

Mivel a FreeBSD Projekt nem regisztrálta magát, a hozzá tartozó érték tehát a `-//FreeBSD` lesz. Látható egyébként, hogy a W3C sem regisztrálta még magát.

Kulcsszó
Az állományban található információ típusának megadására különböző kulcsszavak használhatóak. Ezek általában a `DTD`, `ELEMENT`, `ENTITIES` és `TEXT`. A `DTD` csak DTD állományokhoz használatos, az `ELEMENT` pedig olyan DTD részletekhez, amelyekeben csak egyedek vagy elemek leírásai találhatóak. A `TEXT` SGML tartalmat jelent (szövegeket és jelölőket).

Leírás
Minden egyéb adat, amit az állomány tartalmáról még meg akarunk adni. Tartalmazhat verziószámokat, vagy bármilyen számunkra értelmes és az SGML rendszer számára egyedien azonosítható rövid szöveget.

Nyelv
Kétkarakteres, ISO szabvány szerint megadott kód, amellyel az állomány natív nyelvét adjuk meg. Az angol esetében ez az `EN`.

3.3.1.1. Katalógusok

Az SGML feldolgozónak a dokumentum feldolgozása során valamilyen módon az így megadott formális publikus azonosítóból vissza kell tudnia fejtenie a DTD-t tartalmazó állomány nevét.

Mindehhez egy katalógusra lesz szükségünk. Ezek (amelyeket általában `catalog` néven találhatunk meg) tartalmazzák az azonosítók és a hozzájuk tartozó állománynevek összerendeléseit. Például, ha egy katalógusban a következő sor található:

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

Az SGML feldolgozó a `catalog` állományt tartalmazó könyvtár `4.0` alkönyvtárában levő `strict.dtd` állományban fogja keresni az adott DTD-t.

Nézzünk szét kicsit a `/usr/local/share/xml/html/catalog` állományban. Ez tartalmazza a textproc/docproj portból telepített HTML DTD-kre vonatkozó állományinformációkat.

3.3.1.2. Az `SGML_CATALOG_FILES` környezeti változó

Az SGML feldolgozónak valahogy tudnia kell hol keresse a katalógusokat. Számos implementációjuk parancssori paramétereken keresztül teszi lehetővé a feldolgozás során használni kívánt katalógus vagy katalógusok felsorolását.

Emellett az `SGML_CATALOG_FILES` környezeti változó segítségével is megadhatóak ezek az állományok. A változó értékének a (teljes elérési útvonallal kifejtett) katalógusok vesszővel tagolt felsorolását kell beállítani.

Az esetek döntő többségében a következő állományokat kell ilyen módon felvennünk:

- `/usr/local/share/xml/docbook/4.1/catalog`
- `/usr/local/share/xml/html/catalog`
- `/usr/local/share/xml/iso8879/catalog`
- `/usr/local/share/xml/jade/catalog`

Ezt egyébként korábban már elvégeztük.

3.3.2. Azonosítók helyett

A formális publikus azonosítók használata helyett akár közvetlenül meg is adhatjuk a dokumentum érvényességét definiáló DTD-t tartalmazó konkrét állomány nevét.

Ennek felírása némileg eltér:

```
<!DOCTYPE html SYSTEM -"/az/elérési/út/állomány.dtd">
```

A `SYSTEM` kulcsszóval jelezzük, hogy az SGML feldolgozó a DTD leírását rendszerszinten keresse. Ez általában (de nem mindig) azt jelenti, hogy a DTD elérhetőségét állománynév formájában adjuk meg.

Az FPI használata leginkább a hordozhatóság támogatása miatt ajánlott. Ilyenkor nem kell ugyanis a dokumentumokhoz mellékelni a DTD egy példányát, illetve ha a `SYSTEM` kulcsszóval adnánk meg, akkor mindig az adott helyre kellene tenni a DTD állományokat.

3.4. Visszaváltás az SGML használatára

Az alapismeretek tárgyalása során már korábban szóba került, hogy az SGML csak a dokumentumtípus-deklaráció leírásához használatos. Ez azonban nem tökéletesen igaz, mivel léteznek olyan SGML-beli szerkezetek, amelyeket magukban a dokumentumokban is fel tudunk használni. Például a dokumentumokba beszúrhatóak az elemző részéről figyelmen kívül hagyandó megjegyzések. Ezeket az SGML szabályai szerint adjuk meg. A többi szerkezet használatára kicsivel később mutatunk még példákat.

Nyilvánvalóan valamilyen módon jeleznünk kell az SGML feldolgozónak, hogy a soron következő elem tartalmát hagyja figyelmen kívül, de az elemző ezt alapvetően az SGML alapján észleli.

Az ilyen részeket `<! … >` formában jelöljük. Az elhatárolók között szabályos SGML szerkezetek szerepelhetnek, pontosan olyanok, amelyeket az DTD-k készítésénél is alkalmaznak.

Az előbb bemutatott DOCTYPE deklaráció erre éppen remek példaként szolgálhat.

3.5. Megjegyzések

A megjegyzések tehát SGML-beli konstrukciók, és általában csak a DTD-ben érvényes a használatuk. A 3.4. szakasz - Visszaváltás az SGML használatáraban viszont láthattuk, hogy az SGML szerkezetei akár a dokumentumokban is használhatóak.

Az SGML megjegyzéseket „- -” szimbólumok használatával határolhatjuk el. A szimbólum első előfordulásával kezdjük a megjegyzést és a másodikkal zárjuk le.

3.8. példa - Általános SGML megjegyzés

```
<!-- próba megjegyzés --->

<!-- Most a megjegyzés belsejében vagyunk --->

<!-- Ez pedig egy másik megjegyzés    --->

<!-- Így lehet például
     többsoros megjegyzéseket írni --->

<!-- Ez egy másik módja a   ---
  --- többsoros megjegyzések írásának --->
```

Ha dolgoztunk már korábban HTML kóddal, akkor előfordulhat, hogy más meghatározást láttunk a megjegyzésekre. Ezért tévesen azt gondolhattuk, hogy a megjegyzéseket a `<!--` karaktersorozat vezeti be, és csak a `-->` zárhatja le.

Valójában viszont *nem* így van. Sok böngésző hibás HTML elemzőt tartalmaz, ezért ezt érvényesnek fogadják el. A Dokumentációs Projektben használt SGML elemzők azonban ennél sokkal szigorúbbak és az ilyen hibás dokumentumokat visszadobják.

3.9. példa - Hibás SGML megjegyzések

```
<!-- Most egy megjegyzés belsejében vagyunk --->

     KÍVÜL VAGYUNK A MEGJEGYZÉSEN!

  --- ismét megjegyzésben vagyunk --->
```

Az SGML elemző ezt valahogy így fogja értelmezni:

```
<!KÍVÜL VAGYUNK A MEGJEGYZÉSEN>
```

Ez nem szabályos SGML és ráadásul félrevezető hibaüzenetet eredményez.

```
<!----- Ez nem szép dolog! ------>
```

A példa szövege szerint *sem* javasolt ilyen megjegyzéseket írni.

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

Ez már (valamivel) értelmesebb megoldás, de még feláll a veszélye, hogy megtéveszti az SGML-ben járatlan olvasókat.

3.5.1. Egy kis gyakorlás...

1. Tegyünk néhány megjegyzést a korábban készített `próba.xml` állományunkba, majd az `onsgmls` segítségével ellenőrizzük, hogy közben érvényes marad.

2. Tegyünk néhány érvénytelen megjegyzést a `próba.xml` állományba, majd nézzük meg, hogy az `onsgmls` milyen hibaüzeneteket ad rájuk.

3.6. Egyedek

Az egyedek felhasználásával neveket tudunk rendelni a tartalom egyes darabjaihoz. Az SGML elemző a dokumentum feldolgozása közben ezeket az egyedeket megtalálja és helyükre beszúrja az általuk hivatkozott tartalmat.

Ezzel az SGML dokumentumokban könnyedén ki tudunk alakítani újrafelhasználható, gyorsan cserélhető tartalmat, illetve kizárólag ezen a módon lehet jelölőkkel ellátott SGML állományokat beletenni egy másik hasonló SGML állományba.

Az egyedek kétfajta típusa létezik, amelyek mindegyike eltérő helyzetekben használható: ezek az *általános egyedek* és a *paraméteregyedek*.

3.6.1. Általános egyedek

Általános egyedeket nem lehet SGML környezetben használni (habár definiálni igen), egyedül magában a dokumentumban. Vessük össze a paraméteregyedekkel.

Mindegyik általános egyed rendelkezik egy névvel. Így tudunk hivatkozni egy általános egyedre (ezáltal mindarra a szövegre, amelyet képvisel a dokumentumunkban): `&egyednév;`. Vegyük például, hogy van egy `jelenlegi.valtozat` nevű egyedünk, amely a termékünk jelenlegi verziószámát helyettesíti be a szövegbe. Ezt így fogalmaznánk meg:

```
<para>A termékünk jelenlegi változata a(z)
  &jelenlegi.valtozat;.</para>
```

Így a verziószám változásakor egyszerűen csupán az általános egyed definícióját kell megváltoztatni, majd újra feldolgozni a dokumentumot.

Az általános egyedek segítségével olyan karakterek is megadhatóvá válnak, amelyeket egyébként nem tudnánk SGML dokumentumokban leírni. Például a < és a & normális esetben nem lehet része SGML dokumentumoknak. Amikor ugyanis az SGML elemző egy < szimbólumot észlel, feltételezi, hogy ezzel egy (kezdő- vagy záró) címke kezdődik, illetve amikor pedig egy & szimbólumot talál, akkor a következő lépésben egy egyed nevét várja.

Szerencsére ezek a szimbólumok a szövegben bármikor kiválthatóak az `<` és `&` általános egyedek használatával.

Általános egyedek csak SGML környezetben definiálhatóak. Ezeket általában közvetlenül a DOCTYPE deklaráció után sorolják fel.

3.10. példa - Általános egyedek definíciója

```
<!DOCTYPE html PUBLIC -"-//W3C//DTD HTML 4.0//EN" [
<!ENTITY jelenlegi.valtozat    -"3.0-RELEASE">
<!ENTITY legutolso.valtozat    -"2.2.7-RELEASE">
]>
```

Figyeljük meg, hogy a DOCTYPE deklarációt a sor végén egy szögletes nyitó zárójel elhelyezésével kibővítettük: a kiegészítésként felvett két egyedet az utána következő két sorban definiáltuk, majd bezártuk a szögletes zárójelet és DOCTYPE deklarációt.

A szögletes zárójelek szükségesek ahhoz, hogy jelezzük a DOCTYPE deklarációban megadott DTD további kiegészítéseit.

3.6.2. Paraméteregyedek

Az általános egyedekhez hasonlóan a paraméteregyedek is újrafelhasználható szövegrészek elnevezését engedik meg. Miközben azonban az általános egyedek csak a dokumentumokban alkalmazhatóak, addig a paraméteregyedek csak SGML környezetekben használhatóak.

A paraméteregyedek az általános egyedekhez hasonló módon definiálhatóak, az `&egyednév;` felírás helyett azonban az `%egyednév;` alakban tudunk rájuk hivatkozni. Továbbá a definíciójukban az `ENTITY` kulcsszó és az egyed neve közé be kell szúrni a `%` (százalékjel) szimbólumot.

3.11. példa - Paraméteregyedek megadása

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

```
<!ENTITY % param.valami -"valami">
<!ENTITY % param.szoveg -"szöveg">
<!ENTITY % param.uj  -"%param.valami más %param.szoveg">
```

Ez most még nem tűnik különösebben hasznosnak. Később viszont majd az lesz.

3.6.3. Egy kis gyakorlás...

1. Tegyük bele a `próba.xml` állományunkba a következő általános egyedet:

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

<html>
  <head>
    <title>Egy minta HTML állomány</title>
  </head>

  <body>
    <p>Ebben a bekezdésben van egy kis szöveg.</p>

    <p>Ez a bekezdés még tartalmaz némi szöveget.</p>

    <p align="right">Ennek a bekezdésnek jobbra zártnak kellene ↺
lennie.</p>

    <p>A dokumentum jelenlegi változata: &valtozat;</p>
  </body>
</html>
```

2. Az `onsgmls` használatával vizsgáltassuk meg a dokumentum érvényességét.

3. Töltsük be a `próba.xml` állományt a böngészőnkbe (előfordulhat, hogy másolatot kell készíteni róla `próba.html` néven, mert a böngészőnk csak így ismerné fel HTML dokumentumként).

   Hacsak a böngészőnk nem annyira fejlett, a dokumentumban a `&valtozat;` egyedhivatkozás nem fog lecserélődni a verziószámra. A böngészők többségében nagyon primitív elemzők találhatóak, amelyek nem képesek rendesen kezelni az SGML dokumentumokat[1].

4. A megoldást a dokumentum *normalizálása* jelenti, amelyet egy SGML normalizálóval tudunk elvégezni. A normalizáló beolvas egy érvényes SGML állományt és

[1] Micsoda szégyen! Képzeljük csak el, mennyi gondot és ügyeskedést (mint például a szerver oldalán beemelt állományokat) el tudnánk kerülni, ha rendesen támogatnák.

eredményként egy szintén érvényes, de valamilyen módon átalakított SGML állományt készít. Az SGML állományok átalakításának egyik ilyen módja a dokumentumban található egyedhivatkozások helyettesítése az általuk képviselt szöveggel.

Erre a célra az `osgmlnorm` használható.

```
% osgmlnorm próba.xml > próba.html
```

Ennek hatására `próba.html` néven létrejön a dokumentum normalizált (vagyis a kifejtett egyedhivatkozásokkal létrehozott) változata, és most már betölthető a böngészőnkbe.

5. Ha most megnézzük az `osgmlnorm` által gyártott végeredményt, akkor tapasztalhatjuk, hogy az elején nem szerepel DOCTYPE deklaráció. Ezt a `-d` kapcsolóval tehetjük hozzá:

```
% osgmlnorm --d próba.xml > próba.html
```

3.7. Állományok tartalmának elérése egyedeken keresztül

Az (általános és paraméter-) egyedek különösen hasznosak olyan esetekben, amikor állományok tartalmát akarjuk beilleszteni másik állományokba.

3.7.1. Állományok tartalmának elérése általános egyedekkel

Tegyük fel, hogy egy könyvön dolgozunk az SGML felhasználásával, amelyet fejezetenként állományokra bontottunk, `fejezet1.xml` , `fejezet2.xml` stb. néven, illetve a `könyv.xml` állomány tartalmazza ezeket a fejezeket.

Az állományok tartalmát a `SYSTEM` kulcsszó használatával tudjuk egyedek értékeként megadni. Ennek hatására az SGML elemző a megadott állomány tartalmát adja az egyed értékének.

3.12. példa - Állományok tartalmának elérése általános egyeddel

```
<!DOCTYPE html PUBLIC -"-//W3C//DTD HTML 4.0//EN" [
<!ENTITY fejezet.1 SYSTEM -"fejezet1.xml">
<!ENTITY fejezet.2 SYSTEM -"fejezet2.xml">
<!ENTITY fejezet.3 SYSTEM -"fejezet3.xml">
]>
```

```
<html>

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

Figyelem

Amikor általános egyedeken keresztül illesztünk be állományokat egy másik állományba, a beillesztett állományok (amilyen például a `fejezet1.xml` , `fejezet2.xml` és a többi) *nem* kezdődhetnek DOCTYPE deklarációval. Ez szintaktikai hibát eredményez!

3.7.2. Állományok tartalmának elérése paraméteregyedekkel

Emlékezzünk vissza, hogy a paraméteregyedek csak SGML környezetben alkalmazhatóak. Miért akarnánk állományokat beilleszteni egy SGML környezetbe?

Így tudunk gondoskodni az általános egyedek újrafelhasználhatóságáról.

Tegyük fel, hogy a dokumentumunkban rengeteg fejezet található és ezeket két különböző könyvben is felhasználtuk, azonban eltérő stílusban.

A könyvek elején fel lehetne sorolni az egyedeket, de ezzel viszont gyorsan kezelhetetlenné válnának.

Ehelyett csak tegyünk az általános egyedekre vonatkozó definíciókat egyetlen állományba és a dokumentumunkban erre építve paraméteregyedek beiktatásával végezzük az adott állományok beillesztését.

3.13. példa - Állományok beillesztése paraméteregyedekkel

Először vegyük az egyedek definícióit egy külön `fejezetek.ent` állományba. Ebben a következőek találhatóak:

```
<!ENTITY fejezet.1 SYSTEM -"fejezet1.xml">
<!ENTITY fejezet.2 SYSTEM -"fejezet2.xml">
<!ENTITY fejezet.3 SYSTEM -"fejezet3.xml">
```

Most pedig hozunk létre egy paraméteregyedet az állomány tartalmának hivatkozására. Ezután az iménti paraméteregyeddel illesszük be az állományt a dokumentumba, így az összes általános egyed elérhetővé válik. Innentől már a megszokott módon használhatjuk az általános egyedeket:

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

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

3.7.3. Egy kis gyakorlás...

3.7.3.1. Állományok beillesztése általános egyedek segítségével

1. Hozzunk létre három állományt: `bekezd1.xml` , `bekezd2.xml` és `bekezd3.xml` .

   Töltsük fel ezeket valami hasonló szöveggel:

   ```
   <p>Ez az első bekezdés.</p>
   ```

2. Szerkesszük át a `próba.xml` állományunk tartalmát az alábbi módon:

   ```
   <!DOCTYPE html PUBLIC -"-//W3C//DTD HTML 4.0//EN" [
   <!ENTITY valtozat -"1.1">
   <!ENTITY bekezd1 SYSTEM -"bekezd1.xml">
   <!ENTITY bekezd2 SYSTEM -"bekezd2.xml">
   <!ENTITY bekezd3 SYSTEM -"bekezd3.xml">
   ]>

   <html>
     <head>
       <title>Próba HTML állomány</title>
     </head>

     <body>
       <p>A dokumentum jelenlegi változata: &valtozat;</p>

       &bekezd1;
       &bekezd2;
       &bekezd3;
     </body>
   </html>
   ```

3. A `próba.xml` normalizálásával hozzuk létre a `próba.html` állományt.

```
% osgmlnorm --d próba.xml > próba.html
```

4. Nyissuk meg a böngészőnkkel a `próba.html` állományt és ellenőrizzük, hogy a `bekezdn.xml` állományok tartalma bekerült a `próba.html` állományba.

3.7.3.2. Állományok beillesztése paraméteregyedek segítségével

Megjegyzés

Ehhez először végezzük el az előbbi lépéseket.

1. Szerkesszük át a `próba.xml` állományt a következőeknek megfelelően:

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

<html>
  <head>
    <title>Próba HTML állomány</title>
  </head>

  <body>
    <p>A dokumentum jelenlegi változata: &valtozat;</p>

    &bekezd1;
    &bekezd2;
    &bekezd3;
  </body>
</html>
```

2. Hozzunk létre egy új állományt `egyedek.xml` néven a következő tartalommal:

```
<!ENTITY valtozat -"1.1">
<!ENTITY bekezd1 SYSTEM -"bekezd1.xml">
<!ENTITY bekezd2 SYSTEM -"bekezd2.xml">
<!ENTITY bekezd3 SYSTEM -"bekezd3.xml">
```

3. A `próba.xml` normalizálásával állítsuk elő a `próba.html` állományt:

```
% osgmlnorm --d próba.xml > próba.html
```

4. Nyissuk meg a böngészőnkben a `próba.html` állományt és ellenőrizzük, hogy a `bekezdn.xml` állományok szerepelnek a `próba.html` állományban.

3.8. Jelölt szakaszok

Az SGML tartalmaz olyan megoldást, amellyel a dokumentum bizonyos részeit speciális feldolgozásra jelölhetjük meg. Ezeket nevezik „jelölt szakaszoknak”.

3.14. példa - A jelölt szakaszok felépítése

```
<![ KULCSSZÓ [
  A jelölt szakasz tartalma.
]]>
```

A korábbiakban tapasztaltak szerint természetesen a jelölt szakaszokat az SGML részeként a `<!` szimbólummal vezetjük be.

Ezt követően az első szögletes zárójel határolja el a jelölt szakaszt.

Az elemző a feldolgozás során a *`KULCSSZÓ`* alapján értelmezi az adott jelölt szakaszt.

A második szögletes zárójellel kezdődik a jelölt szakasz tényleges tartalma.

A jelölt szakasz az iménti két szögletes zárójel lezárásával ér véget, majd a `>` szimbólummal visszaváltunk az SGML környezetből a dokumentum környezetébe.

3.8.1. A jelölt szakaszok kulcsszavai

3.8.1.1. `CDATA`, `RCDATA`

A kulcsszavakkal a jelölt szakasz *tartalmi modelljét* tudjuk megváltoztatni.

Az SGML elemző a dokumentum feldolgozása során tárol egy ún. „tartalmi modellt”.

Röviden úgy foglalhatnánk össze, hogy ez a tartalmi modell írja le az elemző részéről várt információkat és azok feldolgozását.

A két ilyen leghasznosabb tartalmi modell a `CDATA` és az `RCDATA`.

A `CDATA` jelentése „Character Data”, vagyis „karakteres adat”. Az elemző ebben a tartalmi modellben kizárólag csak karaktereket lát. Ebben a modellben a `<` és `&` szimbólumok elveszítik különleges jelentésüket.

Az `RCDATA` jelentése „Entity references and character data”, vagyis „egyedhivatkozások és karakteres adatok”. Ebben a tartalmi modellben az elemző karakterekre *és* egyedekre számít. A `<` szimbólum ilyenkor elveszíti a különleges jelentését, azonban az `&` továbbra is általános egyedek kezdetét fogja jelölni.

Ezek használata különösen hasznos abban az esetben, amikor rengeteg < és & karaktert tartalmazó nyers szöveget akarunk beilleszteni valahova a dokumentumba. Természetesen ez megoldható úgy is, ha minden < szimbólumot `<` karaktersorozattá, illetve minden & szimbólumot `&` karaktersorozattá alakítunk, de sokkal könnyebb ezeket a szakaszokat `CDATA` típusúnak megjelölni. Az SGML elemzők ilyenkor tehát figyelmen kívül hagyják a tartalomban talált < és & szimbólumokat.

Megjegyzés

A `CDATA` vagy `RCDATA` kulcsszavakat bemutató SGML példákkal kapcsolatban megjegyezzük, hogy a `CDATA` szakaszok tartalma nem érvényesítődik. Az így beillesztett SGML szöveget valamilyen más módon kell ellenőrizni. Például írjuk meg a karakteres szakasz tartalmát egy másik dokumentumban, ellenőriztessük le, majd másoljuk be a `CDATA` részbe.

3.15. példa - CDATA típusú jelölt szakaszok használata

```
<para>Ebben a példában láthatjuk hogyan tudunk sok ↺
<literal>&lt;</literal>
  és <literal>&amp;</literal> szimbólumot tartalmazó szöveget ↺
elhelyezni
  a dokumentumunkban.  A minta most egy HTML kódrészlet lesz, ↺
az ezt övező
  szöveg (<para>) és (<programlisting>) pedig DocBook.</para>

<programlisting>
  <![CDATA[  ↺
    <p>Ezzel a példával mutatjuk HTML elemek használatát a
      dokumentumban.  Mivel elég sok relációjelet kell ↺
ilyenkor megadni,
      sokkal egyszerűbb azt mondani, hogy legyen az egész ↺
példa egy
      CDATA szakaszban, mintsem végig egyedekkel jelöljük a ↺
balra és
      jobb nyitó relációjeleket.</p>

    <ul>
      <li>Ez egy listaelem</li>
      <li>Ez egy másik listaelem</li>
      <li>Ez már egy harmadik listaelem</li>
    </ul>

    <p>Itt a vége a példának.</p>
  -]]>
```

```
</programlisting>
```

Ha megnézzük a dokumentum forrását, láthatjuk a jelölésnél alkalmazott megoldásokat.

3.8.1.2. `INCLUDE` és `IGNORE`

Az `INCLUDE` kulcsszó megadásakor a jelölt szakasz teljes tartalma feldolgozódik. Ezzel szemben viszont az `IGNORE` kulcsszó esetén a jelölt szakasz tartalmát figyelmen kívül fogja hagyni az elemző és ezáltal nem dolgozódik fel, tehát nem jelenik meg az eredményben.

3.16. példa - Az `INCLUDE` és `IGNORE` használata jelölt szakaszokban

```
<![ INCLUDE [
  Ez a szöveg feldolgozódik és beillesztődik.
]]>

<![ IGNORE [
  Ez a szöveg nem dolgozódik fel és nem is illesztődik be.
]]>
```

Ezek önmagukban nem túlzottan hasznosak, elvégre, ha el akarunk távolítani egy szövegrészt a dokumentumunkból, akkor vagy egyszerűen kivágjuk, vagy megjegyzésbe tesszük.

Sokkal hasznosabbá válhatnak viszont a számunkra, ha észrevesszük, hogy paraméteregyedek segítségével mindez vezérelhető. Emlékezzünk vissza, hogy a paraméteregyedek csak SGML környezetben használhatóak, és a jelölt szakaszokhoz tartozó kulcsszavak *pontosan* egy ilyen SGML környezetben vannak.

Például tegyük fel, hogy egy dokumentáció nyomtatott és elektronikus változatán dolgozunk egyszerre. Az elektronikus változatban szeretnénk azonban néhány olyan elemet is betenni, amelyeket nem akarunk megjelentetni nyomtatásban.

Hozzunk létre egy paraméteregyedet és legyen az értéke `INCLUDE`. Készítsük el a dokumentumot, és jelölt szakaszokkal határoljuk el a csak az elektronikus változat megjelenő részeket. Ezekben a jelölt szakaszokban a kulcsszavak helyére írjuk be az előbbi paraméteregyedet.

Amikor a dokumentumot nyomtatásra akarjuk előkészíteni, akkor legyen a paraméteregyed értéke `IGNORE`, majd dolgozzuk fel újra az egész dokumentumot.

3.17. példa - Jelölt szakaszok vezérlése paraméteregyeddel

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

...

<![ %elektronikus.valtozat [
  Ez a rész csak a dokumentum elektronikus változatában fog ↺
megjelenni.
]]>
```

A nyomtatott változat előkészítésekor így állítsuk át az egyed értékét:

```
<!ENTITY % elektronikus.valtozat -"IGNORE">
```

A dokumentum újbóli feldolgozása során a jelölt szakaszok a `%elektronikus.valtozat` értékét fogják kulcsszóként megkapni, és így kimaradnak.

3.8.2. Egy kis gyakorlás...

1. A következő szöveggel hozzunk létre egy állományt `szakasz.xml` néven:

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

<html>
  <head>
    <title>Példa a jelölt szakaszok használatára</title>
  </head>

  <body>
    <p>Ez a bekezdés <![CDATA[sok-sok <
      karaktert (< < < < <) tartalmaz, így érdemesebb
      CDATA szakaszba tenni]]>.</p>

    <![IGNORE[
    <p>Ez a bekezdés egyértelműen nem fog látszódni az ↺
eredményben.</p>
    -]]>

    <![ %szoveges.kimenet [
```

```
    <p>Ez a bekezdés nem fog feltétlenül megjelenni az ↺
eredményben.</p>

    <p>A konkrét megjelenését a %szoveges.kimenet
      paraméteregyed értéke befolyásolja.</p>
    -]]>
  </body>
</html>
```

2. A `osgmlnorm` használatával normalizáljuk ezt az állományt, majd elemezzük az eredményt. Nézzük meg melyik bekezdések tűntek el, melyek jelentek meg és mi történt a CDATA szakaszok tartalmával.

3. A `szoveges.kimenet` értéke legyen `INCLUDE` az `IGNORE` helyett. Futassuk le újra így a normalizálást és vizsgáljuk meg mi változott az eredményben.

3.9. Befejezés

Ezzel befejeztük az SGML alapismeretek bemutatását. A helyigény, illetve a bonyolultság visszaszorítása érdekében bizonyos témákkal teljes mélységében (vagy egyáltalán) nem foglalkoztunk, azonban az iménti szakaszokban elkerült SGML ismeretek elegendőek lesznek az FDP által készített dokumentáció megértéséhez.

4. fejezet - Az SGML alkalmazása

Ebben a fejezetben a FreeBSD Dokumentációs Projekt keretein belül két leggyakrabban előforduló jelölőnyelvet ismerhetjük meg. Az egyes szakaszokban ezen nyelvek bemutatására, illetve jelenleg alkalmazott vagy alkalmazandó jelölési sajátosságaira térünk ki.

Az itt tárgyalt jelölőnyelvek nagy számú elemet tartalmaznak, és ezért gyakorta zavarba ejtő lehet az adott helyzetnek leginkább megfelelő elemek kiválasztása a rengeteg kínálkozó alternatíva közül. Ebben a szakaszban ezért igyekezük érinteni az összes fontosabb elemet, valamint példákat mutatni a megfelelő használatukra.

Ez az összefoglalás természetesen *nem* tartalmazza mindegyik elemet, mivel ezzel lényegében a nyelv saját dokumentációját írnánk le ismételten. Ebben a szakaszban elsősorban inkább azon elemek ismertetését tűztük ki célul, amelyek a munkánk során valószínűleg a leghasznosabbaknak fognak bizonyulni. A további különböző jelölési megoldásokra vonatkozóan bátran kérhetünk tanácsot a FreeBSD Dokumentációs Projekt levelezési lista tagjaitól!

Belső elemek kontra blokkok

A leírás további részeiben *belsőnek* nevezzük azokat az elemeket, amelyek szerepelhetnek blokkelemekben és nem okoznak sortörést. Ezzel szemben viszont a *blokk* formátumú elemek feldolgozása sortörést (vagy egyéb feldolgozási lépéseket) eredményez.

4.1. HTML

A HTML, más néven HyperText Markup Language, a Világháló jelölőnyelve. Ezzel kapcsolatban részlesebb leírásokat a `http://www.w3.org/` címen találhatunk.

A HTML használata a FreeBSD honlapján található oldalak készítésénél jelenik meg. Más dokumentációkhoz azonban (általánosságban) nem szokták alkalmazni, mivel a DocBook ennél sokkal bőségesebb eszközöket kínál fel. Ennek következményeképpen tehát többnyire csak a honlap fejlesztése során fogunk HTML oldalakkal találkozni.

A HTML létrejötte óta több verzióváltáson is keresztülment már, az 1, 2, 3.0, 3.2 verziókat követően egészen a legfrissebb 4.0 változatáig (amely egyaránt elérhető *szigorú* (strict) és *enyhébb* (loose) formáiban is).

A HTML DTD-k a Portgyűjteményből a textproc/html porton keresztül érhetőek el. A textproc/docproj port ezt automatikusan telepíti.

4.1.1. Formális publikus azonosító

A HTML megfelelni kívánt verziójától (amelyet sokszor szintnek is szoktak nevezni) függően különböző formális publikus azonosító (FPI) áll rendelkezésünkre.

A FreeBSD honlapján található HTML dokumentumok többsége a HTML 4.0 enyhébb változatának felel meg:

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

4.1.2. A dokumentum részei

A HTML dokumentumok általános esetben két részre oszthatóak. Az első, *fejnek* nevezett rész tartalmazza a dokumentumhoz tartozó metainformációkat, például a címét, a szerző nevét, a szülődokumentumot és így tovább. A második, *törzsnek* hívott rész pedig a felhasználó részére megjelenített tartalmat foglalja magában.

A dokumentum ezen részeit rendre a `head` (mint angolul a „fej”) és a `body` (mint angolul a „törzs”) elemekkel jelöljük. Ezeket az elemeket végül a legfelsőbb szinten álló `html` elem tartalmazza.

4.1. példa - Egy átlagos HTML dokumentum felépítése

```
<html>
  <head>
  <title>A dokumentum címe </title>
  </head>

  <body>

    …

  </body>
</html>
```

4.1.3. Blokkok

4.1.3.1. Fejlécek

A HTML lehetővé teszi fejlécek jelölését egészen hat különböző szintig.

A legnagyobb és legkiemelkedőbb fejléc a `h1`, majd ezt követi a `h2`, egészen a `h6` címkéig.

Az elem tartalma a fejléc szövege lesz.

4.2. példa - A `h1`, `h2`, stb. elmek

A használat módja:

```
<h1>Első szakasz</h1>

<!-- Ide kerül a dokumentum bevezető része --->

<h2>Ez az első szakasz fejléce</h2>

<!-- Az első szakasz tartalma következik --->

<h3>Ez az első alszakasz fejléce</h3>

<!-- Az első alszakasz tartalma itt foglal helyet --->

<h2>Ez a második szakasz fejléce</h2>

<!-- A második szakasz tartalma ezen a részen lesz --->
```

A HTML oldalaknak általában rendelkezniük kell első szintű fejléccel (`h1`). Ez tetszőleges számú második szintű fejlécet (`h2`) tartalmazhat, amelyek szintén tetszőleges mennyiségű harmadik szintű fejlécet. Ügyeljünk arra, hogy minden h*n* elem mindig a nála eggyel nagyobb szintű elemet tartalmazza, a sorszámozásban tehát nem javasolt közöket hagyni.

4.3. példa - A `hn` elemek helytelen sorrendje

A használati módja:

```
<h1>Első szakasz</h1>

<!--> A dokumentum bevezetője --->
```

```
<h3>Alszakasz</h3>

<!-- Nem jó, mert kimaradt a <h2> szint! --->
```

4.1.3.2. Bekezdések

A HTML egyetlen bekezdésfajtát ismer, ez a `p`.

4.4. példa - A `p` elem

A használat módja:

```
<p>Ez egy bekezdés.  Szinte bármilyen más elemet ↺
tartalmazhat.</p>
```

4.1.3.3. Idézetblokkok

Az idézetblokkok más dokumentumok nagyobb részeinek idézésére használhatóak az aktuális bekezdésen túl.

4.5. példa - A `blockquote` elem

A használat módja:

```
<p>Részlet a Szózatból:</p>

<blockquote>Hazádnak rendületlenűl
  Légy híve, oh magyar,
  Bölcsőd az -'s majdan sírod is,
  Melly ápol -'s eltakar.

  A' nagy világon e' kivűl
  Nincsen számodra hely,
  Áldjon vagy verjen sors' keze,
  Itt élned, halnod kell.

  Ez a' föld, mellyen annyiszor
  Apáid' vére folyt;
  Ez, mellyhez minden szent nevet
  Egy ezredév csatolt.

  Itt küzdtenek honért a' hős
```

```
  Árpádnak hadai,
  Itt törtek össze rabigát
  Hunyadnak karjai.

  Szabadság! Itten hordozák
  Véres zászlóidat,
  -'S elhulltanak legjobbjaink
  A' hosszu harcz alatt.</blockquote>
```

4.1.3.4. Felsorolások

A dokumentumokban háromféle felsorolást használhatunk: sorszámozott, sorszámozás nélkül és definíciós.

Röviden úgy mutathatnánk be ezeket a formátumokat, hogy a sorszámozott felsorolásban az elemek elé számok kerülnek, a sorszámozás nélküli esetben pontok, a definíciós felsorolásban pedig a bejegyzések két részéből jönnek létre az elemek. Ezek közül az első részben a meghatározandó fogalom található, míg a második részben annak meghatározása.

A sorszámozott felsorolásokat az `ol` elem jelzi, a sorszámozás nélküli felsorolásokat az `ul` elem, végül a definíciós felsorolásokat a `dl` elem.

A sorszámozott és sorszámozás nélküli felsorolások a felsorolás elemeit tartalmazzák, amelyeket a `li` elemekkel vezetünk be. A felsorolások elemeinek szöveges tartalma lehet, vagy ha ezeket becsomagoljuk egy vagy több `p` elembe, további elemeket tartalmazhatnak.

A definíciós felsorolások meghatározandó fogalmakat (`dt`) és meghatározásokat (`dd`) tartalmazhatnak. A meghatározandó fogalmat tartalmazó részben csak belső elemek szerepelhetnek. A meghatározásokban viszont további blokkok is megjelenhetnek.

4.6. példa - Az `ul` és `ol` elemek

A használat módja:

```
<p>Egy sorszámozás nélkül felsorolás.  A felsorolás elemei ↺
előtt minden
  bizonnyal pontok fognak megjelenni.</p>

<ul>
  <li>Első elem</li>

  <li>Második elem</li>

  <li>Harmadik elem</li>
```

```
</ul>

<p>Egy sorszámozott lista, ahol az elemek több bekezdésből ↺
állnak.
  Mindegyik elem (figyelem: nem mindegyik bekezdés) előtt egy ↺
sorszámnak kell
  szerepelnie.</p>

<ol>
  <li>
    <p>Ez az első elem.  Ennek csak egy bekezdése van.</p>
  </li>

  <li>
    <p>Ez a második elem első bekezdése.</p>

    <p>Ez a második elem második bekezdése.</p>
  </li>

  <li>
    <p>Ez az első és egyetlen bekezdés a harmadik elemben.</p>
  </li>
</ol>
```

4.7. példa - Definíciós felsorolások a `dl` elemmel

A használat módja:

```
<dl>
  <dt>Első fogalom</dt>

  <dd>
    <p>Az első fogalom meghatározásának első bekezdése.</p>

    <p>Az első fogalom meghatározásának második bekezdése.</p>
  </dd>

  <dt>Második fogalom</dt>

  <dd>
    <p>A második fogalom meghatározásának első bekezdése.</p>
  </dd>

  <dt>Harmadik fogalom</dt>

  <dd>
    <p>A harmadik fogalom meghatározásának első bekezdése.</p>
  </dd>
```

```
</dl>
```

4.1.3.5. Formázott szöveg

Megadhatjuk, hogy a szöveg egyes részei pontosan abban a formában kerüljenek a felhasználó elé, ahogy az eredetileg szerepel. Ilyenkor általában a szöveg rögzített szélességű betűtípussal jelenik meg, az egymás mellett levő szóközök nem vonódnak össze, a sortörések hatása fontossá válik.

Mindezt a `pre` elemen keresztül érhetjük el.

4.8. példa - A `pre` elem

A `pre` elem például remekül alkalmas e-mailek jelölésére:

```
<pre>  From: Gabor PALI &lt;pgj@FreeBSD.org&gt;
  To: bsd@hu.FreeBSD.org
  Subject: Uj FreeBSD-cikk forditas: Naplozo UFS hasznalata ↺
asztali szamitogepeken

  Kedves listatagok!


    Nemreg elkeszitettem az ``Implementing UFS Journaling on ↺
a Desktop
  PC'' neven szerepelo [1] FreeBSD-cikk magyar forditasat [2].
  Szeretnek megkerni mindenkit, akit erdekel a honositott ↺
valtozat,
  hogy olvassa el, nezze at, betatesztelje es mondjon rola ↺
velemenyt.
  Egyelore meg csak a sajat Perforce repositorynkbol erheto ↺
el, de a
  megadott linken naponta egyszer automatikusan frissul a ↺
HTML valtozat
  a feltoltott valtoztatasok (peldaul hibajavitasok) ↺
fuggvenyeben.
  Elore is nagyon szepen koszonom mindenkinek a segitseget!

  -:g

  [1] http://www.freebsd.org/doc/en/articles/gjournal-desktop/
  [2] http://people.freebsd.org/~pgj/gjournal-desktop_hu/</pre>
```

Hasznos azonban tudnunk, hogy a `<` és `&` jelek a formázott szövegben továbbra is speciális jelentéssel bírnak. A példában ezért is használtunk `<` egyedeket a `<` jelek helyett. Ugyanezért a `>` a `>` helyén is látható. Ezért mindig

körültekintően bánjunk a nyers szövegből, például e-mailből vagy forráskódból bemásolt részletekkel, és ne felejtsük el átalakítani a bennük található speciális karaktereket.

4.1.3.6. Táblázatok

Megjegyzés

A legtöbb (Lynx-hez hasonló) szöveges módban futó böngésző kifejezetten ügyetlen módon jeleníti meg a táblázatokat. Ha az oldalon a táblázatos felépítést választjuk, akkor a problémák elkerüléséhez érdemes egy alternatív jelölési módszert alkalmazni.

A táblázatos formában megjeleníteni kívánt információt jelöljük a `table` elemmel. A táblázatok egy több sorból (`tr` mint „table row”) állnak, amelyek egy vagy több adatcellát (`td` mint „table data”) tartalmaznak. Mindegyik cella tartalmazhat további blokkokat, például bekezdéseket vagy listákat, de akár táblázatokat (ez a beágyazás tetszőleges mélységig folytatható). Ha a cella tartalma csak egyetlen bekezdés, akkor nincs szükség a `p` elem használatára.

4.9. példa - A `table` egyszerű használata

A használat módja:

```
<p>Ez egy 2x2-es táblázat.</p>

<table>
  <tr>
    <td>Bal felső cella</td>

    <td>Jobb felső cella</td>
  </tr>

  <tr>
    <td>Bal alsó cella</td>

    <td>Jobb alsó cella</td>
  </tr>
</table>
```

Egy cella több sorra vagy oszlopra is átnyúlhat. Ennek jelzéséhez a kiterjesztendő sorokhoz a `rowspan` és/vagy oszlopokhoz a `colspan` tulajdonságot adjuk meg a megfelelő értékkel.

4.10. példa - A `rowspan` tulajdonság

A használat módja:

```
<p>Egy magas keskeny cella a bal oldalon, mellette jobbra
  két rövid cella.</p>

<table>
  <tr>
    <td rowspan="2">Hosszú és keskeny</td>
  </tr>

  <tr>
    <td>Felső cella</td>

    <td>Alsó cella</td>
  </tr>
</table>
```

4.11. példa - A `colspan` tulajdonság

A használat módja:

```
<p>Felül egy hosszú cella, alatt két rövidebb cella.</p>

<table>
  <tr>
    <td colspan="2">Felső cella</td>
  </tr>

  <tr>
    <td>Bal alsó cella</td>

    <td>Jobb alsó cella</td>
  </tr>
</table>
```

4.12. példa - A `rowspan` és `colspan` tulajdonságok együttes használata

A használat módja:

```
<p>Egy 3x3-as rácson a bal felső blokk 2x2 egymásba olvasztott
  cellából áll.  A többi cella normális.</p>

<table>
  <tr>
    <td colspan="2" rowspan="2">Bal felső nagy cella</td>

    <td>Jobb felső cella</td>
  </tr>

  <tr>
    <td>Jobb középső cella</td>
  </tr>

  <tr>
    <td>Bal alsó cella</td>

    <td>Bal középső cella</td>

    <td>Jobb alsó cella</td>
  </tr>
</table>
```

4.1.4. Belső elemek

4.1.4.1. Az információ kiemelése

A HTML esetén a kiemelésnek két szintje létezik, az `em` és a `strong`. Ezek közül az `em` jelenti a hagyományos kiemelést és a `strong` az erősebbet.

Az `em` elem tartalma általában dőlt betűvel jelenik meg, miközben a `strong` elem tartalma félkövéren. Ez a megállapítás azonban nem minden esetben igaz, ezért nem szabad semmi ilyesmit feltételeznünk a használatukkor.

4.13. példa - A `em` és `strong` elemek

A használat módja:

```
<p><em>Ezt</em> a részt kiemeltük, miközben <strong>ezt</
strong> részt
  erősebben kiemeltük.</p>
```

4.1.4.2. Félkövér és dőlt formázás

Mivel a HTML tartalmaz konkrétan a megjelenítésre vonatkozó jelölőket is, ezért külön jelezni tudjuk a forrásban, hogy a szöveg melyik részét szeretnénk félkövéren vagy dőlten látni. Ezeket a funkciókat a `b`, illetve az `i` elemekkel érhetjük el.

4.14. példa - A `b` és `i` elemek

```
<p><b>Ez</b> félkövér, <i>ez</i> pedig dőlt.</p>
```

4.1.4.3. Írógépszerű formázás

Az írógépszerűen (rögzített szélességű karakterekkel) írt szövegek formázásához a `tt` (mint „teletype”) elemet használhatjuk.

4.15. példa - A `tt` elem

A használat módja:

```
<p>Ezt a dokumentumot eredetileg Páli Gábor fordította,
  és a következő címen érhető el: <tt>pgj@FreeBSD.org</tt>.</p>
```

4.1.4.4. Méretezés

Előfordulhat, hogy szeretnénk növelni vagy csökkenteni a szöveg megjelenítéséhez használt betűtípus méretét. Erre alapvetően három lehetőség kínálkozik.

1. Ágyazzuk az átméterezendő szöveget `big` és `small` elemekbe. Ezeket a címkéket tetszőleges mélységig egymásba tudjuk ágyazni, tehát írható olyan, hogy `<big><big>Ez már sokkal nagyobb!</big></big>` .

2. Használjuk a `font` elemet, és a `size` tulajdonságát állítsuk a `+1` vagy `-1` értékre. Ez hatása szerint megegyezik a `big` és a `small` elemek használatával, azonban ez a típusú megoldás már elavult.

3. A `font size` tulajdonsága `1` és `7` között állítható. A betű alapértelmezett mérete `3`. Ez a megközelítés már elavult.

4.16. példa - A `big`, `small` és a `font` elemek

A következő kódrészleteknek ugyanaz a hatása.

```
<p>Ez a szöveg <small>valamivel kisebb</small>.  Ez a
  a szöveg viszont <big>valamivel nagyobb</big>.</p>

<p>Ez a szöveg <font size="-1">valamivel kisebb</font>.  Ez a
  szöveg viszont <font size="+1">valamivel nagyobb</font>.</p>

<p>Ez a szöveg <font size="2">valamivel kisebb</font>.  Ez a
  szöveg viszont <font size="4">valamivel nagyobb</font>.</p>
```

4.1.5. Hivatkozások

Megjegyzés

A hivatkozások is belső elemek.

4.1.5.1. Hivatkozás más dokumentumokra a Világhálón

A Világhálón elhelyezkedő dokumentumokat úgy tudjuk hivatkozni, ha ismerjük a helyüket.

A hivatkozást az `a` elemmel adhatjuk meg, amelynek a `href` tulajdonsága tartalmazza a hivatkozott dokumentum helyét. Az elem tartalma ekkor egy hivatkozássá változik, és a felhasználó számára is jól látható módon jelenik meg (aláhúzással, más színnel, más egérmutatóval és így tovább).

4.17. példa - Az `<a href="...">` elem

A használat módja:

```
<p>Erre vonatkozóan részletesebb információkat a
```

```
  <a href="http://www.FreeBSD.org/">FreeBSD honlapján</a> ↺
találhatunk.</p>
```

Ezeket a hivatkozások a felhasználót az adott dokumentum elejére irányítják.

4.1.5.2. A dokumentumok más részeinek hivatkozása

A dokumentumok egyéb részeire (akár ugyanazon a dokumentumon belül) csak akkor tudunk hivatkozni, ha a szerző előzetesen hivatkozási pontokat helyeztünk el bennük.

Hivatkozási pontokat szintén `a` elemekkel adhatunk meg, azonban a `href` tulajdonság helyett a `name` használatával.

4.18. példa - Az `<a name="...">` elem

A használat módja:

```
<p><a name="bekezd1">Ez</a> a bekezdés a hivatkozásokban a ↺
<tt>bekezd1</tt>
  névvel érhető el.</p>
```

A dokumentum így megnevezett részét egy egyszerű hivatkozás készítésével tudjuk elérni, azonban a hivatkozási pont neve elé tennünk kell egy # jelet.

4.19. példa - Egy másik dokumentum nevesített részének elérése

Tételezzük fel, hogy a `bekezd1` példánk az `ize.html` állományban található.

```
<p>A témáról további információkat az <tt>ize.html</tt>
  <a href="ize.html#bekezd1">első bekezdésében</a> ↺
találhatunk.</p>
```

Ha egy hivatkozási pontra ugyanazon a dokumentumon belül szeretnénk hivatkozni, akkor nyugodtan elhagyhatjuk a dokumentum nevét, elegendő egyszerűen magát a hivatkozási pontot megadnunk (természetesen a hozzá tartozó # jellel együtt).

4.20. példa - Ugyanazon dokumentum nevesített részének elérése

Tételezzük fel, hogy a `bekezd1` példa ugyanezen a dokumentumon belül helyezkedik el:

```
<p>A témáról további információkat az
  <a href="#bekezd1">első bekezdésben</a> találhatunk.</p>
```

4.2. DocBook

A DocBook jelölőnyelvet eredetileg a HaL Computer Systems és az O'Reilly & Associates dolgozta ki a műszaki jellegű dokumentációk írásához[1], illetve 1998 óta a DocBook Műszaki Bizottság tartja karban. Mint ilyen nyelv, eltérően a LinuxDoc és a HTML megoldásaitól, a DocBook erősen olyan jelölések irányába orientálódik, amelyek nem azt írják le *hogyan*, hanem *mit* jelenítsünk meg.

`Formális` kontra **`informális`**

Bizonyos elemeknek létezik ún. *formális* és *informális* változata. A formális változat általában egy címből és az adott elem informális változatából áll. Az informális változat nem tartalmaz címet.

A DocBook használatához szükséges DTD a Portgyűjteményből a textproc/docbook porton keresztül érhető el. Ez a textproc/docproj port részeként automatikusan telepítődik.

4.2.1. A FreeBSD kiterjesztései

A FreeBSD Dokumentációs Projekt kiterjesztette a hivatalos DocBook DTD-t további elemekkel. Segítségükkel bizonyos jelölések sokkal pontosabbá tehetőek.

A kizárólag a FreeBSD esetén alkalmazott elemeket egyértelműen jelezni fogjuk a felsorolásban.

[1] Ennek rövid története a `http://www.oasis-open.org/docbook/intro.shtml#d0e41` címen olvasható.

A dokumentum további részében a „DocBook” kifejezés a DocBook DTD FreeBSD kiterjesztéseivel együtt értendő.

Megjegyzés

Szeretnénk megemlíteni, hogy a hozzáadott kiegészítésekben azonban semmi olyan nincs, ami kizárólag a FreeBSD-re vonatkozna, egyszerűen csak a Projektben felmerült igények mentén szeretne alkalmazni néhány javítást. Ha más UNIX® jellegű rendszerek (NetBSD, OpenBSD, Linux, stb.) fejlesztőit esetleg érdekelné a DocBook további fejlesztése, kérjük, vegyék fel velünk a kapcsolatot a Documentation Engineering Team <doceng@FreeBSD.org > címén.

A FreeBSD kapcsán alkalmazott kiterjesztések (jelenleg) nem érhetőek el a Portgyűjteményből, hanem a FreeBSD repositoryban találjuk meg a doc/share/xml/freebsd.dtd helyen.

4.2.2. Formális publikus azonosító

A DocBook a testreszabott változatok formális publikus azonosítóira vonatkozó irányelvei szerint a FreeBSD kiterjesztéseivel bővített DocBook DTD formális publikus azonosítója a következő lesz:

```
PUBLIC -"-//FreeBSD//DTD DocBook V4.1-Based Extension//EN"
```

4.2.3. A dokumentum szerkezete

A DocBook többféle módon kínál lehetőségeket a dokumentumok szerkezetének kialakítására. A FreeBSD Dokumentációs Projektben a DocBook dokumentumok két alapvető fajtáját használjuk, a könyvet és a cikket.

A könyvek fejezetekből (`chapter`) állnak, amelyek használata kötelező. A könyv és a fejezetek közé még további szervezési rétegként beilleszthetőek részek (`part`) is. Például a FreeBSD kézikönyv szerkezetét is ennek megfelelően alakítottuk ki.

A fejezetek tartalmazhatnak (vagy sem) egy vagy több szakaszt, amelyeket `sect1` elemekkel jelezhetünk. Amennyiben egy szakasz újabb szakaszt tartalmaz, akkor használjuk a `sect2` elemet, és így tovább egészen a `sect5` szintig.

A fejezetek és a szakaszok tartalmazzák a dokumentum tartalmának fennmaradó részét.

Egy cikk egy könyvnél egyszerűbb felépítésű, és nem tartalmaz fejezeteket. Helyette a cikkek tartalmát egy vagy több szakaszba szervezzük, a könyvnél már említett `sect1` (`sect2` és így tovább) elemek segítségével.

A készítendő dokumentációról értelemszerűen jellegének mérlegelésével tudjuk eldönteni, hogy könyvként esetleg cikk-ként érdemesebb jelölni. A cikk formátum választása leginkább olyan információk esetén célszerű, ahol nincs szükségünk külön fejezetekre. Röviden szólva tehát egy viszonylag rövid, legfeljebb 20-25 oldalas írást takar. A könyv formátum ezzel szemben leginkább olyan esetekben alkalmazható, amikor az információ fejezetekre bontható, amelyhez függelékek és hasonlók is társulhatnak.

A FreeBSD-hez készített cikkek mindegyikét cikk-ként jelöltük, miközben például ez a dokumentum, a FreeBSD GYIK, és a FreeBSD kézikönyv könyvként került jelölésre.

4.2.3.1. Könyv írása

A könyvek tartalmát egy `book` elemben adjuk meg. Ez a jelölő amellett, hogy magában foglalja a könyv teljes felépítését, tovább információkat tud tárolni magáról a könyvről. Ez lehet akár hivatkozási célokat szolgáló metainformáció, vagy éppen a címlap elkészítéséhez szükséges egyéb leírás.

A könyvre vonatkozó további információkat egy `bookinfo` elemen belül adhatjuk meg.

4.21. példa - Egy `book` és `bookinfo` elemek segítségével definiált könyvsablon

```
<book>
  <bookinfo>
    <title>Ide írjuk a címet </title>

    <author>
      <surname>Vezetéknév </surname>
      <firstname>Keresztnév </firstname>
      <affiliation>
        <address><email>E-mail cím </email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2008</year>
      <holder role="mailto:E-mail cím ">Név</holder>
    </copyright>

    <releaseinfo>$FreeBSD$</releaseinfo>

    <abstract>
      <para>Ide kerüljön a könyv tartalmának rövid ↺
összefoglalása. </para>
```

```
    </abstract>
  </bookinfo>

  …

</book>
```

4.2.3.2. Cikk írása

A cikk tartalma az `article` elembe kerül. A dokumentum szervezésén kívül ennek az elemnek feladata lehetőséget kínálni további információk elhelyezésére. Ez lehet hivatkozási célokra alkalmas metainformáció, vagy például a címlap előállításához szükséges egyéb adatok.

A cikk-kel kapcsolatos további információk egy `articleinfo` elemben adhatóak meg.

4.22. példa - Egy `article` és `articleinfo` elemek segítségével definiált cikksablon

```
<article>
  <articleinfo>
    <title>Ide írjuk a címet </title>

    <author>
      <surname>Vezetéknév </surname>
      <firstname>Keresztnév </firstname>
      <affiliation>
        <address><email>E-mail cím </email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2008</year>
      <holder role="mailto:E-mail cím ">Név</holder>
    </copyright>

    <releaseinfo>$FreeBSD$</releaseinfo>

    <abstract>
      <para>Ide kerüljön a cikk tartalmának rövid ↺
összefoglalása. </para>
    </abstract>
  </articleinfo>

  …
```

```
</article>
```

4.2.3.3. Fejezetek készítése

A `chapter` elem használatával tudunk fejezeteket jelölni. Minden fejezetnek kötelezően rendelkeznie kell egy címmel, vagyis egy `title` elemmel. A cikkek nem tartalmazhatnak fejezeteket, kizárólag könyvek számára tartják fenn.

4.23. példa - Egy egyszerű fejezet

```
<chapter>
  <title>Fejezetcím</title>

  -...
</chapter>
```

A fejezetek nem lehetnek üresek, a `title` elem mellett még tartalmazniuk kell valamilyen másik elemet is. Az üres fejezetek készítéséhez használjunk egy üres bekezdést.

4.24. példa - Üres fejezetek

```
<chapter>
  <title>Ez egy üres fejezet</title>

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

4.2.3.4. Szakaszok fejezetek alatt

A könyvekben a fejezetek további szakaszokra, alszakaszokra stb. bonthatóak (de nem kötelező). A cikkekben azonban a szakaszok az alapvető szervezőelemek, ezért minden cikknek legalább egy szakaszt tartalmaznia kell. A szakaszok létrehozására a `sect`*n* elemet használhatjuk, ahol az *n* szám adja meg a szakasz szintjét.

Az első ilyen `sect`*n* elem a `sect1`, amelyből egy fejezetben egy vagy több is szerepelhet. Ezek egy vagy több `sect2` elemet tartalmazhatnak, és így tovább egészen az `sect5` szintjéig.

4.25. példa - Szakaszok fejezetekben

```
<chapter>
  <title>Minta fejezet</title>

  <para>Egy kis fejezetbeli szöveg.</para>

  <sect1>
    <title>Első szakasz (1.1)</title>

    &hellip;
  </sect1>

  <sect1>
    <title>Második szakasz (1.2)</title>

    <sect2>
      <title>Első alszakasz (1.2.1)</title>

      <sect3>
        <title>Első al-alszakasz (1.2.1.1)</title>

        &hellip;
      </sect3>
    </sect2>

    <sect2>
      <title>Második alszakasz (1.2.2)</title>

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

Megjegyzés

Láthatjuk, hogy ebben a példában a szakaszok neveiben megjelenik a szakaszok számozása. Ezt azonban ne írjuk bele a dokumentumainkba! A szakaszok számozását a stíluslapok végzik (erről még később szó lesz), ezekkel egyáltalán nem kell foglalkoznunk.

4.2.3.5. A dokumentum felosztása `part` elemek használatával

A `book` és `chapter` elemek részéről felkínált szervezési szintek közé a `part` elemek alkalmazásával egy újabbat tudunk illeszteni. Erre a cikkek esetében nincs lehetőségünk.

```
<part>
  <title>Bevezetés</title>

  <chapter>
    <title>Áttekintés</title>

    -...
  </chapter>

  <chapter>
    <title>Mi a FreeBSD?</title>

    -...
  </chapter>

  <chapter>
    <title>Történet</title>

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

4.2.4. Blokkok

4.2.4.1. Bekezdések

A DocBookban a bekezdések háromféle típusát találhatjuk meg: `formalpara` , `para` és `simpara`.

Az iméntiek közül a legtöbb esetben az `para` elemre lesz szükségünk. A `formalpara` tartalmaz még egy `title` elemet, illetve a `simpara` nem engedélyezi bizonyos elemek használatát a bekezdésben. Érdemes tehát inkább következetesen a `para` használatánál maradni.

> 4.26. példa - A **`para`** elem
>
> A használat módja:
>
> ```
> <para>Ez egy bekezdés.  Tetszőleges egyéb elem megjelenhet ↺
> benne.</para>
> ```
>
> Így jelenik meg:

Ez egy bekezdés. Tetszőleges egyéb elem megjelenhet benne.

4.2.4.2. Idézetblokkok

Az idézetblokkok egy másik dokumentumból származó, kiterjedtebb hosszúságú idézetet jelölnek, amelyeknek az aktuális bekezdéstől függetlenül kell megjelenniük. Erre valószínűleg csak nagyon ritkán lesz ténylegesen szükségünk.

Az idézetblokkok opcionálisan címeket és szerzőt is tartalmazhatnak (de akár szerepelhetnek cím vagy szerző nélkül).

4.27. példa - A `blockquote` elem

A használat módja:

```
<para>Részlet Szerb Antal <quote>A Pendragon legenda</quote> ↺
című
  művéből:</para>

<blockquote>
  <title>A Pendragon legenda</title>

  <attribution>Szerb Antal</attribution>

  <para>Minden nőben azt élveztem, hogy a szimbóluma volt ↺
valaminek.  Volt
    nő, akit azért szerettem, mert ő volt Svédország, volt ↺
nő, akit azért,
    mert a XVIII.  századra emlékeztetett törékeny Sèvres-
mivolta.
    Volt, akiben Jeanne d'Arc-ot álmodtam, volt, akiben az ↺
ezermellű
    ephesusi Dianát.  Cynthiát, ha megcsókoltam, úgy éreztem, ↺
most az
    angol szonetekkel flörtölök, ötödfeles jambusokban.  ↺
Volt, akinek édes
    tehénszerűségében svájci, alpesi réteket élveztem.</para>
</blockquote>
```

Így jelenik meg:

Részlet Szerb Antal „A Pendragon legenda” című művéből:

> A Pendragon legenda
>
> Minden nőben azt élveztem, hogy a szimbóluma volt valaminek. Volt nő, akit azért szerettem, mert ő volt

> Svédország, volt nő, akit azért, mert a XVIII. századra emlékeztetett törékeny Sèvres-mivolta. Volt, akiben Jeanne d'Arc-ot álmodtam, volt, akiben az ezermellű ephesusi Dianát. Cynthiát, ha megcsókoltam, úgy éreztem, most az angol szonetekkel flörtölök, ötödfeles jambusokban. Volt, akinek édes tehénszerűségében svájci, alpesi réteket élveztem.
>
> —Szerb Antal

4.2.4.3. Tanácsok, megjegyzések, felhívások, figyelmeztetések, fontos és mellékes információk

Esetenként szükségünk lehet arra, hogy bizonyos többletinformációt közöljünk az olvasóval a szövegtől elkülöníthető módon. Ez többnyire olyan „metainformáció”, amelyre a felhasználónak tekintettel érdemes lennie.

Az adott információ jellegétől függően erre a célra a `tip` (tanács), `note` (megjegyzés), `warning` (felhívás), `caution` (figyelmeztetés) és `important` (fontos információ) elemek valamelyikét tudjuk használni. Amennyiben a közölni kívánt információ kötődik ugyan a szöveghez, azonban az előbbiek közül egyik kategóriába sem sorolható be, akkor használhatjuk a `sidebar` (mellékinformáció) elemet is.

Nem teljesen tisztázott, hogy az imént felsorolt elemek közül pontosan mikor melyiket kell alkalmazni. A DocBook dokumentációja ezzel kapcsolatban a következőket javasolja:

- A `note` elemek olyan információkat jelölnek, amelyek az olvasó részéről megszivlelendőek.
- Az `important` elemek a `note` elemek egyik változata.
- A `caution` elemmel olyan információt jelölnek, amelyek ismeretének hiányában adatvesztés vagy szoftveres károdás következhet be.
- A `warning` elemek olyan információkat jelölnek, amelyek ismeretének hiánya hardver károsodását, életveszélyt vagy a végtagok sérülését eredményezheti.

4.28. példa - A `warning` elem

A használat módja:

```
<warning>
  <para>A FreeBSD telepítése után könnyen előfordulhat, hogy ↺
a Windowst
```

```
    teljesen le akarjuk törölni a merevlemezünkről.</para>
</warning>
```

Így jelenik meg:

Figyelem

A FreeBSD telepítése után könnyen előfordulhat, hogy a Windowst teljesen le akarjuk törölni a merevlemezünkről.

4.2.4.4. Felsorolások és eljárások

Gyakran adódhatnak olyan helyzetek, amikor az olvasó felé fel kell sorolnunk valamilyen információkat, vagy egy adott cél elérése érdekében be kell mutatnunk neki egy sorszámozott, egyenként végrehajtandó lépéssorozatot.

Erre a célra rendelkezésünkre állnak az `itemizedlist` , az `orderedlist` , illetve a `procedure` elemek[2].

Az `itemizedlist` és az `orderedlist` elemek hasonlóak az HTML esetén már megismert megfelelőikhez, az `ul` és `ol` elemekhez. Egy vagy több `listitem` elemből állnak és mindegyik `listitem` egy vagy több blokkot tartalmaz. A `listitem` elemek a HTML `li` elemeihez hasonlóak, azonban ebben az esetben kötelező megadni ezeket.

A `procedure` elem ettől némileg eltér. Itt `step` elemekből épül fel, amelyek további `step` vagy `substep` típusú elemeket foglalhatnak magukban. A `step` elemek mindegyike blokkokat tartalmaz.

4.29. példa - Az `itemizedlist` , `orderedlist` és `procedure` elemek

A használat módja:

```
<itemizedlist>
  <listitem>
    <para>Ez a felsorolás első eleme.</para>
  </listitem>

  <listitem>
```

[2] A felsorolások megadására a DocBook további lehetőségeket is felkínál, azonban ezekkel itt most nem foglalkozunk.

```
    <para>A a felsorolás második eleme.</para>
  </listitem>
</itemizedlist>

<orderedlist>
  <listitem>
    <para>Ez az első sorszámozott elem.</para>
  </listitem>

  <listitem>
    <para>Ez a második sorszámozott elem.</para>
  </listitem>
</orderedlist>

<procedure>
  <step>
    <para>Csináljuk ezt.</para>
  </step>

  <step>
    <para>Majd csináljuk azt.</para>
  </step>

  <step>
    <para>Most pedig csináljuk így.</para>
  </step>
</procedure>
```

Így jelenik meg:

- Ez a felsorolás első eleme.
- Ez a felsorolás második eleme.

1. Ez az első sorszámozott elem.
2. Ez a második sorszámozott elem.

1. Csináljuk ezt.
2. Majd csináljuk azt.
3. Most pedig csináljuk így.

4.2.4.5. Példák állományokra

Ha állományrészeket (vagy akár teljes állományokat) akarunk bemutatni az olvasónak, akkor érdemes ezeket egy `programlisting` elembe illeszteni.

A `programlisting` elemeken belül alkalmazott tördelés és a sortörések helye *jelentéssel bír*. Ennek egyik fontos következménye, hogy a nyitócímkének az állomány tartalmának első sorával együtt kell megjelennie, illetve a zárócímkének pedig az utolsó sorban, ellenkező esetben üres sorok fognak keletkezni a kimenetben.

4.30. példa - A `programlisting` elem

A használat módja:

```
<para>Miután befejeztük a feladatot, a programunknak valahogy ↺
így kell
  majd kinéznie:</para>

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

int
main(void)
{
    printf("Halló mindenki!\n");
}</programlisting>
```

Láthatjuk, hogy az `#include` után a relációs jeleket nem közvetlenül adtuk meg, hanem a nekik megfelelő egyedekkel.

Így jelenik meg:

Miután befejeztük a feladatot, a programunknak valahogy így kell majd kinéznie:

```
#include <stdio.h>

int
main(void)
{
    printf("Halló mindenki!\n");
}
```

4.2.4.6. Magyarázó szövegek

A magyarázó szövegek használatával a szöveg egy korábbi részére vagy egy korábbi példa adott helyére tudunk visszahivatkozni anélkül, hogy a szövegen magán belül hivatkoznánk rá.

Ehhez a co elem használatával jelöljük be a példa (`programlisting` , `literallayout` vagy bármi más) fontosabb részeit. Mindegyik elemhez egy egyedi azonosítót kell társítanunk.

A példa után helyezzünk el egy `calloutlist` elemet, amelyben a megfelelő további magyarázat megadásával együtt visszahivatkozunk a példa egyes részeire.

4.31. példa - A `co` és `calloutlist` elemek

```
<para>Miután befejeztük a feladatot, a programunknak valahogy ↺
így kell
  majd kinéznie:</para>

<programlisting>#include &lt;stdio.h&gt; <co id="co-ex-
include"/>

int <co id="co-ex-return"/>
main(void)
{
    printf("Halló mindenki!\n"); <co id="co-ex-printf"/>
}</programlisting>

<calloutlist>
  <callout arearefs="co-ex-include">
    <para>A szabványos állományműveleteket tartalmazó header
      állomány.</para>
  </callout>

  <callout arearefs="co-ex-return">
    <para>Megadjuk, hogy a <function>main()</function> ↺
függvény egy
      <literal>int</literal> típusú értékkel térjen vissza.</
para>
  </callout>

  <callout arearefs="co-ex-printf">
    <para>A <function>printf()</function> hívással egy
      <literal>Halló mindenki!</literal> szöveget írunk ki a ↺
szabványos
      kimenetre.</para>
  </callout>
</calloutlist>
```

Így jelenik meg:

Miután befejeztük a feladatot, a programunknak valahogy így kell majd kinéznie:

```
#include <stdio.h> ❶

int ❷
main(void)
{
    printf("Halló mindenki!\n"); ❸
```

```
}
```

❶ A szabványos állományműveleteket tartalmazó header állomány.
❷ Megadjuk, hogy a `main()` függvény egy `int` típusú értékkel térjen vissza.
❸ A `printf()` hívással egy `Halló mindenki!` szöveget írunk ki a szabványos kimenetre.

4.2.4.7. Táblázatok

A HTML kapcsán megismertektől eltérően a szöveg elrendezésének befolyásolásához nem kell táblázatokat használnunk, mivel erről majd a stíluslapok gondoskodnak helyettünk. Ehelyett egyszerűen csak akkor használjunk táblázatokat, amikor táblázatosan akarunk adatokat megadni.

Általános értelemben véve (ennek további részleteit lásd a DocBook leírásában) a táblázatok (amelyek lehetnek formálisak vagy informálisak) egy `table` elemből állnak. Ennek magában kell foglalnia legalább egy csoportot jelölő `tgroup` elemet, amely (tulajdonságként) megadja, hogy benne mennyi oszlop található. Ezekben a csoportokban aztán a `thead` elemmel fejlécet adhatunk meg az egyes oszlopoknak, illetve azok törzseit pedig `tbody` elemek specifikálják.

A `tgroup` és `thead` elemek egyaránt tartalmaznak `row` elemeket, amelyek pedig `entry` elemekre bonthatóak tovább. Mindegyik ilyen `entry` elem a táblázat egy celláját jelöli.

4.32. példa - Az `informaltable` elem

A használat módja:

```
<informaltable pgwide="1">
  <tgroup cols="2">
    <thead>
      <row>
        <entry>Ez az első oszlop fejléce</entry>
        <entry>Ez a második oszlop fejléce</entry>
      </row>
    </thead>

    <tbody>
      <row>
        <entry>Első oszlop, első sor</entry>
        <entry>Második oszlop, első sor</entry>
      </row>

      <row>
        <entry>Első oszlop, második sor</entry>
```

```
        <entry>Második oszlop, második sor</entry>
      </row>
    </tbody>
  </tgroup>
</informaltable>
```

Így jelenik meg:

Ez az első oszlop fejléce	Ez a második oszlop fejléce
Első oszlop, első sor	Második oszlop, első sor
Első oszlop, második sor	Második oszlop, második sor

Az `informaltable` elemek esetén a `pgwide` tulajdonságot mindig az `1` értékkel használjuk, ellenkező esetben az Internet Explorer egyik hibája miatt a táblázat nem fog rendesen megjelenni.

Amennyiben nem szeretnénk keretet a táblázathoz, állítsuk az `informaltable` elem `frame` tulajdonságát a `none` értékre (vagyis `<informaltable frame="none">`).

4.33. példa - A `frame="none"` típusú táblázat

Így jelenik meg:

Ez az első oszlop fejléce	Ez a második oszlop fejléce
Első oszlop, első sor	Második oszlop, első sor
Első oszlop, második sor	Második oszlop, második sor

4.2.4.8. Példák műveletekre

Rengetegszer előfordulhat, hogy az olvasónak valamilyen módon be kell mutatnunk miként kell egy bizonyos feladatot megoldania a rendszerben. Ezek a példák általában valamilyen párbeszédet jelentek a számítógéppel: az olvasó begépel egy parancsot, amelyre kap egy választ, majd begépel egy újabb parancsot és így tovább.

Ilyen helyzetek több különböző elem és egyed alkalmazható.

A `screen` elem
: Az olvasó a példával kapcsolatos információkat a elsősorban képernyőről kapja, ennek jelölésére használjuk a `screen` elemet.

  A `screen` elemeken belül számít a szöveg tördelése.

A `prompt` elem, `&prompt.root;` és `&prompt.user;` egyedek
: Az olvasó a képernyőn mindig valamilyen promptot is látni fog (ez lehet az operációs rendszer, a parancsértelmező vagy az adott alkalmazás). Ezt a `prompt` elemmel tudjuk jelölni.

  Az egyszerű felhasználó és a rendszergazda számára két külön egyed létezik a parancssori promptok jelölésére. Amikor tehát az olvasónak egy paranccsorban kell tevékenykednie, akkor a `&prompt.root;` vagy a `&prompt.user;` egyedek valamelyikét használjuk. Ezeket nem kell `prompt` elembe tenni.

Megjegyzés

A `&prompt.root;` és `&prompt.user;` egyedek nem részei az eredeti DocBook DTD-nek, csupán a FreeBSD kiterjesztései.

A `userinput` elem
: A példában az olvasó által begépelendő részeket tegyük `userinput` elemekbe. Ez az olvasó felé valószínűleg majd másképpen jelenik meg.

4.34. példa - A `screen`, `prompt` és `userinput` elemek

A használat módja:

```
<screen>&prompt.user; <userinput>ls --1</userinput>
ize1
ize2
ize3
&prompt.user; <userinput>ls --1 -| grep ize2</userinput>
ize2
&prompt.user; <userinput>su</userinput>
<prompt>Password: </prompt>
&prompt.root; <userinput>cat ize2</userinput>
Ez lenne az -'ize2' nevű állomány.</screen>
```

Így jelenik meg:

```
% ls --1
ize1
```

```
ize2
ize3
% ls --1 -| grep ize2
ize2
% su
Password:
# cat ize2
Ez lenne az -'ize2' nevű állomány.
```

Megjegyzés

Annak ellenére, hogy a példában szerepeltettük az `ize2` állomány tartalmát, *nem* a `programlisting` elemmel jelöltük. A `programlisting` elemeket inkább állományrészletek jelölésében alkalmazzuk, függetlenül az olvasó részéről várt műveletektől.

4.2.5. Belső elemek

4.2.5.1. Az információ kiemelése

Az egyes szavak vagy kifejezések kiemeléséhez alkalmazzuk az `emphasis` elemet. Ennek hatására a kiemelt szövegrész dőlten, esetleg félkövéren jelenik meg, illetve a különböző felolvasó szoftverek más hangsúlyozással mondják el.

A kiemelt szövegrészek tényleges megjelenését nem tudjuk befolyásolni, nem tekinthető egyenlőnek a HTML `b` és `i` elemeivel. Ha fontos információt kívánunk közölni, akkor az `emphasis` helyett érdemesebb inkább az `important` elem használatát megfontolni.

4.35. példa - Az `emphasis` elem

A használat módja:

```
<para>A FreeBSD az Intel architektúrán kétség kívül
  <emphasis>a</emphasis> legjobb UNIX-szerű operációs ↺
rendszer.</para>
```

Így jelenik meg:

A FreeBSD az Intel architektúrán kétség kívül *a* legjobb UNIX-szerű operációs rendszer.

4.2.5.2. Idézetek

Ha más dokumentumokból vagy forrásból akarunk idézni a szövegben, esetleg valamire csak képletesen szeretnénk utalni, akkor használjuk a `quote` elemet. A `quote` elemen belül a legtöbb jelölő elérhető a szövegben.

4.36. példa - Idézetek

A használat módja:

```
<para>Arra viszont ügyeljünk, hogy hogy a keresési rend ne ↺
lépje át a
  <quote>helyi és nyilvános adminisztráció között meghúzódó
  határt</quote>, ahogy azt az RFC 1535 nevezi.</para>
```

Így jelenik meg:

Arra viszont ügyeljünk, hogy hogy a keresési rend ne lépje át a „helyi és nyilvános adminisztráció között meghúzódó határt”, ahogy azt az RFC 1535 nevezi.

4.2.5.3. Billentyűk, egérgombok és azok kombinációja

A billentyűzeten egy adott billentyűre a `keycap` elem segítségével tudunk hivatkozni. Ugyanezt a lehetőséget az egér gombjaira vonatkozóan a `mousebutton` elem nyújta. A billentyűk és egérgombok együttes használatát pedig a `keycombo` elemmel tudjuk jelölni.

A `keycombo` elemnek van egy `action` (tevékenység) nevű tulajdonsága, amely lehet `click` (kattintás), `double-click` (kettős kattintás), `other` (egyéb), `press` (nyomva tartás), `seq` (egymás utáni), illetve `simul` (együttes használat). Az utóbbi két értékkel jelölhetjük, hogy a megadott billentyűket vagy gombokat egymás után esetleg egyszerre kell lenyomnunk.

Az elemben felsorolt billentyűk és gombok nevei közé a stíluslapok automatikusan beillesztik a megfelelő összekapcsoló szimbólumot, például a + jelet.

4.37. példa - Billentyűk, egérgombok és azok kombinációja

A használat módja:

```
<para>A második virtuális terminálra az <keycombo
    action="simul"><keycap>Alt</keycap><keycap>F1</keycap></
keycombo>
```

```
  billentyűkombinációval tudunk
  átváltani.</para>

<para>A <command>vi</command> programból úgy tudunk mentés ↺
nélkül kilépni, ha begépeljük a <keycombo
    action="seq"><keycap>Esc</keycap><keycap>:</keycap>
  <keycap>q</keycap><keycap>!</keycap></keycombo> sorozatot.</
para>

<para>Az ablakkezelőt most úgy állítottuk be, hogy az <keycombo
    action="simul"><keycap>Alt</keycap>
  <mousebutton>jobb</mousebutton></keycombo> egérgomb ↺
segítségével
  tudjuk mozgatni az ablakokat.</para>
```

Így jelenik meg:

A második virtuális terminálra az Alt+F1 billentyűkombinációval tudunk átváltani.

A vi programból úgy tudunk mentés nélkül kilépni, ha begépeljük az Esc : q ! sorozatot.

Az ablakkezelőt most úgy állítottuk be, hogy az Alt+jobb egyérgomb segítségével tudjuk mozgatni az ablakokat.

4.2.5.4. Alkalmazások, parancsok, kapcsolók és man hivatkozások

Nem szokatlan az igény, hogy a dokumentáció írása során gyakran szeretnénk hivatkozni alkalmazásokra és parancsokra egyaránt. A két fajta elem közti különbség egyszerű: az alkalmazás az adott feladatot megvalósító programcsomag (vagy program) neve, miközben a parancs konkrétan annak a programnak a neve, amelyet az olvasó futtatni tud.

Mindezek mellett alkalmanként szükségünk lehet arra, hogy a parancs által várt egy vagy több paramétert valamilyen módon felsoroljuk.

Végül hozzátesszük, hogy sokszor szükségünk lehet a parancsokat a hozzájuk tartozó man oldalakkal együtt hivatkozni, a UNIX típusú kézikönyvek megszokott „parancs(szám)” jelölésben.

Az alkalmazások neveit az `application` elemmel tudjuk jelölni.

Ha egy parancsot a hozzá tartozó man oldallal együtt akarunk hivatkozni (amire valószínűleg az esetek nagy többségében szükségünk is lesz), akkor az ennek megfelelő Docbook elem a `citerefentry` lesz. Ez további két elemet tartalmaz, ezek a `refentrytitle` és a `manvolnum`. A `refentrytitle` tartalma a parancs neve, illetve a `manvolnum` lesz a hozzá tartozó man oldal megfelelő szekciója.

Az iménti jelölések írása esetenként nehézkesnek bizonyulhat, ezért ennek megkönnyítésére létrehoztunk általános egyedeket. Az egyedek `&man.man-oldal.man-szekció;` alakban érhetőek el.

Ezeket az egyedeket a `doc/share/xml/man-refs.ent` állományban találjuk meg, amelyre a következő formális publikus azonosító segítségével tudunk hivatkozni:

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

Ezért tehát a dokumentumunk elején minden bizonnyal szerepelni fog egy ilyen sor:

```
<!DOCTYPE book PUBLIC -"-//FreeBSD//DTD DocBook V4.1-Based ↺
Extension//EN" [

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

…

]>
```

A `command` elemmel a parancsok neveit tudjuk a szövegben hivatkozni közvetlenül, az olvasó által begépelendő formában.

Az `option` elem segítségével a parancsok számára megadható kapcsolókat jelölhetjük.

Amikor többször ugyanarra a parancsra hivatkozunk egymáshoz viszonylag közel, a `&man.parancs.szekció;` típusú jelölést érdemes csak az első hivatkozásnál alkalmazni, a többi inkább legyen egyszerűen csak `command` elemben. Az ebből készített kimenet, különösen a HTML esetében így kinézetében sokkal olvashatóbb.

A jelölési megoldások közti választás ettől függetlenül időnként nem mindig egyértelmű, de remélhetőleg a következő példa segít ebben.

4.38. példa - Alkalmazások, parancsok és kapcsolók

A használat módja:

```
<para>A <application>sendmail</application> az egyik ↺
legelterjedtebb
  levelező alkalmazás UNIX rendszereken.</para>

<para>A <application>sendmail</application> alkalmazás részei a
  <citerefentry>
    <refentrytitle>sendmail</refentrytitle>
```

```
    <manvolnum>8</manvolnum>
  </citerefentry>, &man.mailq.1; és &man.newaliases.1;
  programok.</para>

<para>A <citerefentry>
    <refentrytitle>sendmail</refentrytitle>
    <manvolnum>8</manvolnum>
  </citerefentry> egyik kapcsolója a <option>-bp</option>, ↺
amellyel a
  levelezési sorban található üzenetek aktuális állapotát ↺
kérdezhetjük
  le.  Ezt a <command>sendmail --bp</command> parancs ↺
kiadásával tehetjük
  meg.</para>
```

Így jelenik meg:

A sendmail az egyik legelterjedtebb levelező alkalmazás UNIX rendszereken.

A sendmail alkalmazás részei a sendmail(8), mailq(1) és newaliases(1) programok.

A sendmail(8) egyik kapcsolója a `-bp`, amellyel a levelezési sorban található üzenetek aktuális állapotát kérdezhetjük le. Ezt a `sendmail -bp` parancs kiadásával tehetjük meg.

Megjegyzés

Figyeljük meg, hogy a `&man.parancs.szekció;` jelölés mennyivel könnyebben olvasható.

4.2.5.5. Állományok, könyvtárak, kiterjesztések

Amikor állományok neveire, könyvtárakra, esetleg kiterjesztésekre akarunk hivatkozni, használjunk a `filename` elemeket.

4.39. példa - A `filename` elem

A használat módja:

```
<para>A kézikönyv magyar változatának SGML forrása a <filename
    class="directory">/usr/doc/hu_HU.ISO8859-2/books/handbook/
</filename>
```

```
  könyvtárban található.  Ebben a könyvtárban a
  <filename>book.xml</filename> lesz a fő forrásállomány.  ↺
Mellette
  láthatunk még egy <filename>Makefile</filename> állományt ↺
és több
  <filename>.ent</filename> kiterjesztéssel rendelkező
  állományt.</para>
```

Így jelenik meg:

A kézikönyv magyar változatának SGML forrása a `/usr/doc/hu_HU.ISO8859-2/books/handbook` könyvtárban található. Ebben a könyvtárban a `book.xml` lesz a fő forrásállomány. Mellette találhatunk még egy `Makefile` állományt és több `.ent` kiterjesztéssel rendelkező állományt.

4.2.5.6. A portok nevei

A FreeBSD kiterjesztése

Ezek az elemek a FreeBSD DocBookhoz készített kiterjesztéseinek részei, az eredeti DocBook DTD-ben nem szerepelnek.

Esetenként szükségünk lehet a FreeBSD Portgyűjteményében található bizonyos programok nevének megemlítésére a dokumentációban. Ezt a `filename` elem `role` tulajdonságának a `package` értékre állításával tehetjük meg. Mivel a Portgyűjtemény tetszőleges helyre telepíthető, ezért mindig csak a port kategóriáját és nevét adjuk meg, a `/usr/ports` rész elhagyásával.

4.40. példa - A `filename` elem és a `package` role együttes használata

A használat módja:

```
<para>A hálózati forgalom figyeléséhez telepítsük a <filename
    role="package">net/ethereal</filename> portot.</para>
```

Így jelenik meg:

A hálózati forgalom figyeléséhez telepítsük a net/ethereal portot.

4.2.5.7. Eszközök

A FreeBSD kiterjesztése

Ezek az elemek a FreeBSD DocBookhoz készített kiterjesztéseinek részei, az eredeti DocBook DTD-ben nem szerepelnek.

Az eszközök hivatkozását két módon jelölhetjük. Az eszközre hivatkozhatunk a `/dev` könyvtárban megjelenő neve szerint, vagy pedig a rendszermagbeli neve szerint. Ez utóbbi esetben használjuk a `devicename` jelölést.

Néha előfordulhat, hogy nincs választási lehetőségünk. Bizonyos eszközök, például a hálózati kártyákhoz nem tartozik semmilyen bejegyzés a `/dev` könyvtárban, esetleg az ott megjelenő nevük teljesen eltér.

4.41. példa - A `devicename` elem

A használat módja:

```
<para>A FreeBSD rendszermagjában a <devicename>sio</devicename>
  eszközöket a soros vonali kommunikációra használjuk.  A
  <devicename>sio</devicename> eszközök az idők során több ↺
különböző
  alakban jelentek meg a <filename>/dev</filename> ↺
könyvtárban, például
  <filename>/dev/ttyd0</filename> és <filename>/dev/cuaa0</
filename>
  néven.</para>

<para>Ezzel szemben a hálózati eszközök, mint például az
  <devicename>ed0</devicename> nem jelennek meg a
  <filename>/dev</filename> könyvtárban.</para>

<para>Az MS-DOS rendszerekben az elsődleges hajlékonylemezes ↺
meghajtót
  az <devicename>a:</devicename> néven érhetjük el, miközben ↺
FreeBSD
  alatt ennek a neve <filename>/dev/fd0</filename>.</para>
```

Így jelenik meg:

A FreeBSD rendszermagjában a `sio` eszközöket a soros vonali kommunikációra használjuk. A `sio` eszközök az idők során több különböző alakban jelentek meg a `/dev` könyvtárban, például `/dev/ttyd0` és `/dev/cuaa0` néven.

Ezzel szemben a hálózati eszközök, mint például az `ed0` nem jelennek meg a `/dev` könyvtárban.

Az MS-DOS rendszerekben az elsődleges hajlékonylemezes meghajtót az `a:` néven érhetjük el, miközben FreeBSD alatt ennek a neve `/dev/fd0` .

4.2.5.8. Hálózati nevek, tartományok, IP-címek és így tovább

A FreeBSD kiterjesztése

Ezek az elemek a FreeBSD DocBookhoz készített kiterjesztéseinek részei, az eredeti DocBook DTD-ben nem szerepelnek.

A vonatkozó információ jellegétől függően a hálózatba kapcsolt számítógépek azonosítására szolgáló adatatokat többféle módon is jelölni tudjuk. Minden esetben a `hostid` elemre fogunk támaszkodni, amely `role` tulajdonságával tudjuk megválasztani a jelölt információ típusát.

Nem szerepel a `role` tulajdonság vagy `role="hostname"`
: A `role` tulajdonság megadása nélkül (vagyis az elem `hostid` …`/hostid` alakú) a jelölt információ egy hálózati név, mint például `freefall` vagy `disznohal` . Ugyanezt explicit módon a `role="hostname"` hozzáadásával tudjuk jelölni.

`role="domainname"`
: Az elem tartalma egy tartomány nevét jelöli, mint például `FreeBSD.org` vagy `inf.elte.hu` . Ekkor nincs benne konkrét hálózati név.

`role="fqdn"`
: Az elem tartalma egy teljes hálózati név, amely már tartalmaz tartománynevet és hálózati nevet.

`role="ipaddr"`
: Az elem egy IP-címet jelöl, négy, pontokkal tagolt szám formájában.

`role="ip6addr"`
: Az elem egy IPv6 formátumú IP-címet jelöl.

`role="netmask"`
: Az elem tartalma egy hálózati maszk, amelyet megadhatunk pontokkal elválasztott számokkal, hexadecimális számjegyekkel vagy a `/` szimbólumot követően egy számmal.

`role="mac"`
: Az elemben egy Ethernet MAC-címet adtunk meg, kétjegyű hexadecimális számok kettőspontokkal tagolt sorozataként.

4.42. példa - A `hostid` elem és a különböző role értékek

A használat módja:

```
<para>A gépünk mindig elérhető <hostid>localhost</hostid> ↺
néven,
  amelyhez a <hostid role="ipaddr">127.0.0.1</hostid> IP-cím
  tartozik.</para>

<para>A <hostid role="domainname">FreeBSD.org</hostid> ↺
tartomány több
  különböző gépet foglal magában, többek közt a <hostid
    role="fqdn">freefall.FreeBSD.org</hostid> és <hostid
    role="fqdn">pointyhat.FreeBSD.org</hostid> címeket.</para>

<para>Amikor egy interfészhez IP-álnéveket társítunk (az
  <command>ifconfig</command> paranccsal), akkor ehhez
  <emphasis>mindig</emphasis> a <hostid
    role="netmask">255.255.255.255</hostid> hálózati maszkot ↺
adjuk meg
  (amelyet <hostid role="netmask">0xffffffff</hostid> ↺
formában is
  írhatunk).</para>

<para>A MAC-cím az összes létező hálózati eszközt egyértelműen
  azonosítja.  A MAC-címek általában a <hostid
    role="mac">08:00:20:87:ef:d0</hostid> címhez hasonlóak.</
para>
```

Így jelenik meg:

A gépünk mindig elérhető `localhost` néven, amelyhez a `127.0.0.1` IP-cím tartozik.

A `FreeBSD.org` tartomány több különböző gépet foglal magában, többek közt a `freefall.FreeBSD.org` és `pointyhat.FreeBSD.org` címeket.

Amikor egy interfészhez IP-álnéveket társítunk (az `ifconfig` paranccsal), akkor ehhez *mindig* a `255.255.255.255` hálózati maszkot adjuk meg (amelyet `0xffffffff` formában is írhatunk).

A MAC-cím az összes létező hálózati eszközt egyértelműen azonosítja. A MAC-címek általában a `08:00:20:87:ef:d0` címhez hasonlóak.

4.2.5.9. Felhasználói nevek

A FreeBSD kiterjesztése

Ezek az elemek a FreeBSD DocBookhoz készített kiterjesztéseinek részei, az eredeti DocBook DTD-ben nem szerepelnek.

Ha felhasználókra (például `root` vagy `bin`) kell hivatkoznunk a szövegben, használjuk a `username` elemet.

4.43. példa - A `username` elem

A felhasználás módja:

```
<para>A rendszerünk karbantartásával kapcsolatos legtöbb ↺
feladatot
  kizárólag csak a <username>root</username> felhasználóval ↺
tudjuk
  elvégezni.</para>
```

Így jelenik meg:

A rendszerünk karbantartásával kapcsolatos legtöbb feladatot kizárólag csak a `root` felhasználóval tudjuk elvégezni.

4.2.5.10. A `Makefile` állományokkal kapcsolatos jelölések

A FreeBSD kiterjesztése

Ezek az elemek a FreeBSD DocBookhoz készített kiterjesztéseinek részei, az eredeti DocBook DTD-ben nem szerepelnek.

A `Makefile` állományok egyes részeinek jelöléséhez a `maketarget` és `makevar` elemeket tudjuk használni.

A `maketarget` azokat a `Makefile` állományokban megadott fordítási célokat azonosítja, amelyeket a `make` paramétereként lehet használni. A `makevar` pedig azokat a (környezetben, a `make` hívásakor vagy a `Makefile` állományon belül definiált) változókat azonosítja, amelyekkel a fordítás folyamát lehet szabályozni.

4.44. példa - A `maketarget` és a `makevar` elemek

A használat módja:

```
<para>A <filename>Makefile</filename> állományokban két igen ↺
gyakori cél
  az <maketarget>all</maketarget> és a
  <maketarget>clean</maketarget>.</para>

<para>Az <maketarget>all</maketarget> megadásakor általában
  újrafordítjuk az alkalmazást, a <maketarget>clean</
maketarget>
  megadásakor pedig eltávolítjuk a fordítás közben keletkezett
  ideiglenes állományokat (például az <filename>.o</filename>
  állományokat).</para>

<para>A <maketarget>clean</maketarget> viselkedését számos ↺
változó
  befolyásolja, többek közt a <makevar>CLOBBER</makevar> és a
  <makevar>RECURSE</makevar>.</para>
```

Így jelenik meg:

A `Makefile` állományokban két igen gyakori cél az `all` és a `clean`.

Az `all` megadásakor általában újrafordítjuk az alkalmazást, a `clean` megadásakor pedig eltávolítjuk a fordítás közben keletkezett ideiglenes állományokat (például az `.o` állományokat).

A `clean` viselkedését számos változó befolyásolja, többek közt a `CLOBBER` és a `RECURSE`.

4.2.5.11. Formázatlan szöveg

Sokszor lehet szükségünk „formázatlan” szövegekre a dokumentáció írása közben. Ilyen szöveg jellemző módon egy valamelyik másik állományból átvett részlet, vagy amelyet magából a dokumentációból kell szó szerint átmásolni egy állományba.

Néhány esetben a korábban már bemutatott `programlisting` pontosan elegendő ehhez a feladathoz. Azonban ez a jelölési módszer nem minden esetben megfelelő, különösen olyan helyzetekben, amikor az állomány egy részét magába a bekezdésbe akarjuk tenni.

Ilyen alkalmakkor használjuk a `literal` elemet.

4.45. példa - A `literal` elem

A használat módja:

```
<para>A rendszermag konfigurációs állományában a
  <literal>maxusers 10</literal> sor határozza meg különböző
  rendszerszintű táblázatok méretét, és ezáltal ad egy durva ↺
becslést
  arra, hogy a rendszerünk mennyi bejelentkezést lesz képes ↺
egyszerre
  kezelni.</para>
```

Így jelenik meg:

A rendszermag konfigurációs állományában a `maxusers 10` sor határozza meg különböző rendszerszintű táblázatok méretét, és ezáltal ad egy durva becslést arra, hogy a rendszerünk mennyi bejelentkezést lesz képes egyszerre kezelni.

4.2.5.12. Az olvasó által kötelezően kitöltendő részek jelölése

Minden bizonnyal lesznek olyan részek a dokumentációban, ahol meg szeretnénk mutatni az olvasónak mit kell csinálnia, esetleg hivatkozni akarunk egy állomány nevére vagy egy parancsra stb., viszont nem közvetlenül a megadott nevet kell bemásolnia, hanem önmagától kell kipótolnia egy sémát.

Pontosan ilyen eshetőségekre találták ki a `replaceable` elemet. Más elemeken *belül* használva olyan részeket tudunk vele megjelölni, amelyeket az olvasónak kell kitöltenie.

4.46. példa - A `replaceable` elem

A használat módja:

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

Így jelenik meg:

```
% man parancs
```

A `replaceable` több különböző elemen belül is alkalmazható, egyik ilyen a `literal`. Ebben a példában azt is megmutatjuk, hogy a `replaceable` elembe

ténylegesen csak azt a részt kell tennünk, amelyet *az* olvasónak kell hozzátennie, rajta kívül semmi mást nem kell megváltoztatnia.

A használat módja:

```
<para>A rendszermag konfigurációs állományában a ↺
<literal>maxusers <replaceable>n</replaceable></literal> sor ↺
határozza
  meg különböző rendszerszintű táblázatok méretét, és ezáltal ↺
ad egy
  durva becslést arra, hogy a rendszerünk mennyi bejelentkezést
  lesz képes egyszerre kezelni.</para>

<para>Asztali munkaállomások esetén az <i>n</i> helyére írhatjuk ↺
például a <literal>32</literal>
  értéket.</para>
```

Így jelenik meg:

A rendszermag konfigurációs állományában a `maxusers n` sor határozza meg különböző rendszerszintű táblázatok méretét, és ezáltal ad egy durva becslést arra, hogy a rendszerünk mennyi bejelentkezést lesz képes egyszerre kezelni.

Asztali munkaállomások esetén az *n* helyére írhatjuk például a `32` értéket.

4.2.5.13. Hibaüzenetek idézése

Olykor szükségünk lehet a FreeBSD által jelzett hibák jelölésére. A hibák során keletkező pontos hibaüzeneteket tegyük `errorname` elemekbe.

4.47. példa - Az `errorname` elem

A használat módja:

```
<screen><errorname>Panic: cannot mount root</errorname></
screen>
```

Így jelenik meg:

```
Panic: cannot mount root
```

4.2.6. Képek

Fontos

A dokumentációban a képek használatának támogatása jelen pillanatban még csak kísérleti jellegű. Az itt leírt ismeretek valószínűleg nem fognak változni, de nem szavatoljuk.

A különféle képformátumok közti átalakításokhoz még telepítenünk kell a graphics/ImageMagick portot. Ez egy nagy méretű port és a legtöbb részére nincs is konkrétan szükségünk, viszont jelentősen meg tudja könnyíteni a dolgunkat, amikor `Makefile` állományokkal és egyéb infrastrukturális elemekkel dolgozunk. Ez a port *nem* része a textproc/docproj metaportnak, külön kell egyedileg telepítenünk.

A képek használatára talán a legjobb példát a `doc/en_US.ISO8859-1/articles/vm-design` szolgáltatja. Ha tehát nem értenénk teljesen a szakaszban leírtakat, nézzük meg ezt az állományt és a gyakorlatban is látni fogjuk hogyan kapcsolódnak össze az felhasznált elemek. Ne restelljünk kísérletezgetni a különböző formázási stílusokkal, így láthatjuk miként jelennek meg a jelölt képek a formázott kimenetben.

4.2.6.1. Képformátumok

Jelenleg kétféle képformátum támogatott. A beillesztendő kép jellegétől függ, hogy ezek közül ténylegesen melyiket kell majd használnunk a dokumentumban.

Az alapvetően vektoros szerkezetű képeket, mint például a hálózati kapcsolatokat bemutató diagramokat, idővonalakat és ehhez hasonlókat Encapsulated Postscript formátumban érdemes ábrázolnunk. Gondoskodjunk róla, hogy ezek a képek `.eps` kiterjesztéssel rendelkezzenek.

A raszteres képeket, mint például a képernyő elmentett tartalmát Portable Network Graphic formátumban készítsük el. Figyeljünk rá, hogy az ilyen típusú képek kiterjesztése mindig `.png` legyen.

Ezek tehát *kizárólagosan* azok a formátumok, amelyek bekerülhetnek a repositoryba.

A képekhez mindig válasszuk a megfelelő formátumot, teljesen elfogadott a dokumentációban az EPS és PNG formátumú képek vegyes alkalmazása. A `Makefile` állományok gondoskodni fognak a képek formátumuknak megfelelő szabályos feldolgozásáról. *Ugyanazt a képet a repositoryban ne tároljuk el mind a két formátumban!*

Fontos

A Dokumentációs Projekt előreláthatólag a jövőben majd a vektoros képek ábrázolására a Scalable Vector Graphic (SVG) formátumot fogja használni, azonban jelenleg még nem állnak rendelkezésre olyan SVG szerkesztők, amelyek ezt a gyakorlatban is hatékonnyá tennék.

4.2.6.2. Jelölések

A képek jelölése viszonylag egyszerű. Először is készítsünk egy `mediaobject` elemet. A `mediaobject` elembe ezután további, pontosabban specifikáló objektumokat helyezhetünk el. Most két ilyen elemmel foglakozunk, ezek az `imageobject` és a `textobject`.

Egy `imageobject` és két `textobject` elemet kell megadnunk. Az `imageobject` a beilleszteni kívánt kép nevére fog hivatkozni (kiterjesztés nélkül). A `textobject` elemekben olyan információ szerepel, amelyet az olvasó a kép mellett vagy éppen helyett fog látni a dokumentumban.

Ilyen két esetben fordulat elő:

- A dokumentum HTML változatát olvassuk. Ekkor minden képhez szükség van még egy helyettesítő szövegre, amelyet a kép betöltődésekor láthatunk, vagy amikor az egérmutatót a kép felé visszük.
- A dokumentumot nyers szöveges formátumban olvassuk. Ekkor a kép ASCII karakterekből kirakott változatát kellene látnia az olvasónak.

Egy példán keresztül mindez valószínűleg sokkal könnyebben érthetővé válik. Tegyük tehát most fel, hogy van egy `abra1.png` nevű képünk, amelyet szeretnénk betenni a dokumentumba. Ez a kép egy A betűt ábrázol egy téglalapban. A hozzá tartozó jelölés a következő lesz:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="abra1"> ❶
  </imageobject>

  <textobject>
    <literallayout class="monospaced">+---------------+ ❷
|       A       -|
+---------------+</literallayout>
  </textobject>
```

```
  <textobject>
    <phrase>Egy kép</phrase> ❸
  </textobject>
</mediaobject>
```

❶ Helyezzünk el egy `imagedata` elemet az `imageobject` elembe. A `fileref` tulajdonságban kell megadnunk kiterjesztés nélkül a képhez tartozó állomány nevét. A stíluslapok maguktól megállapítják a neki megfelelő kiterjesztést.

❷ Az első `textobject` elemben szerepelnie kell egy `literallayout` elemnek, ahol a `class` tulajdonság értéke `monospaced` legyen. Itt tudjuk megmutatni milyen jól tudunk ASCII karakterekkel rajzolni. Ezt a dokumentum nyers szöveges változatának előállításakor fogjuk felhasználni.

A `literallayout` elem belsejének első és utolsó sorában megfigyelhetjük, hogy közvetlenül a szöveges ábra mellett kezdődnek, így garantálhatjuk, hogy semmilyen további felesleges szóköz nem jelenik meg a generált változatban.

❸ A második `textobject` elemben egy `phrase` elemnek kell lennie. Ennek tartalma lesz a HTML változatban a képhez tartozó `alt` tulajdonság értéke.

4.2.6.3. A `Makefile` felépítése

A `Makefile` állományokban az `IMAGES` változóban kell felsorolnunk a dokumentumhoz tartozó képeket. Ebben a változóban kell megadnunk a képek *forrását*. Tehát például, ha van három ábránk, név szerint az `abra1.eps` , `abra2.png` és `abra3.png` , akkor ennek megfelelően a `Makefile` állománynak a következő sorokat kellene tartalmaznia:

```
…
IMAGES= abra1.eps abra2.png abra3.png
…
```

vagy

```
…
IMAGES=  abra1.eps
IMAGES+= abra2.png
IMAGES+= abra3.png
…
```

A `Makefile` magától el fogja készíteni a dokumentum lefordításához szükséges képek teljes listáját, nekünk egyedül tehát csak azokat a képeket kell megadnunk, amelyeket *mi* készítettünk.

4.2.6.4. Képek és fejezetek alkönyvtárakban

Nem árt óvatosnak lennünk, amikor a dokumentumunkat kisebb állományokra bontjuk szét (lásd 3.7.1. szakasz - Állományok tartalmának elérése általános egyedekkel) több különböző alkönyvtárban.

Tegyük fel, hogy van egy három fejezetből álló könyvünk, ahol az egyes fejezeteket a saját könyvtáraikban tároljuk: `fejezet1/fejezet.xml` , `fejezet2/fejezet.xml` és

`fejezet3/fejezet.xml` . Ha az egyes fejezetekhez képeket akartunk társítani, akkor javasolt ezeket a fejezetek alkönyvtárába (`fejezet1` , `fejezet2` , `fejezet3`) tennünk.

Ekkor azonban ne felejtsük el, hogy a `Makefile` állomány `IMAGES` változójában *és* az `imagedata` elemekben is a könyvtárak neveivel együtt kell hivatkoznunk a képekre.

Például a `fejezet1/abra.png` kép esetében a `fejezet1/fejezet.xml` állományban a következőt kell megadnunk:

```
<mediaobject>
  <imageobject>
    <imagedata fileref="fejezet1/abra1"> ❶
  </imageobject>

  …

</mediaobject>
```

❶ A könyvtár nevét is meg kell adnunk a `fileref` tulajdonságban.

Az ennek megfelelő `Makefile` állomány tartalma:

```
…
IMAGES=  fejezet1/fejezet1.png
…
```

Ezzel már minden remekül működik.

4.2.7. Hivatkozások

Megjegyzés

A hivatkozások is belső elemek.

4.2.7.1. Hivatkozás ugyazon a dokumentumon belül

A dokumentum belül úgy tudunk hivatkozásokat készíteni, ha egyrészt megadjuk honnan hivatkozunk (tehát szükségünk lesz egy olyan elemre, amelyre az olvasó kattinthat vagy megjelölhető a hivatkozás forrásaként), másrészt megadjuk hova hivatkozunk (tehát a célt).

A DocBook összes eleme rendelkezik egy `id` tulajdonsággal, amelyben az adott elem adott példányához tudunk kapcsolni egy egyedi azonosítót.

Ezt az értéket kell megadnunk a hivatkozás forrásának megjelölésekor.

Általában tehát amikor fejezeteket vagy szakaszokat hivatkozunk, érdemes felvennünk hozzájuk egy `id` tulajdonságot.

4.48. példa - Az `id` tulajdonság fejezeteknél és szakaszoknál

```
<chapter id="fejezet">
  <title>Bevezetés</title>

  <para>Ez a bevezetés.  Ebben szerepel egy szintén ↺
azonosítóval
    rendelkező alszakasz.</para>

  <sect1 id="fejezet1-szakasz1">
    <title>Első alszakasz</title>

    <para>Ez az alszakasz.</para>
  </sect1>
</chapter>
```

A dokumentáció írásakor nyilván ennél beszédesebb azonosítókat lesz majd érdemes kitalálnunk. Mindig ügyeljünk arra, hogy az azonosítóknak egyedieknek kell lenniük a dokumentumban (tehát nem csak az adott állományon, hanem a teljes dokumentum belül). Figyeljük meg hogyan képeztük a példában az alszakasz `id` tulajdonságát a fejezet `id` tulajdonságának értékéből. Ezzel szavatoltuk az azonosító egyediségét.

Ha a dokumentum valamelyik közbenső elemére (jellemzően egy bekezdés vagy egy példa közepére) akarunk hivatkozni, akkor használjuk az `anchor` elemet. Ennek az elemnek nincs tartalma, azonban rendelkezik `id` tulajdonsággal.

4.49. példa - Az `anchor` elem

```
<para>Ebben a bekezdésben elrejtettünk egy <anchor
    id="bekezd">hivatkozás forrását.  Ez a dokumentumban nem ↺
fog
  látszani.</para>
```

Ha a dokumentum egy `id` tulajdonsággal rendelkező részére szeretnénk létrehozni egy hivatkozást (amelyet például kattintással el lehet érni), akkor használjuk az `xref` vagy `link` elemeket.

Mind a két imént említett elemnek van egy `linkend` tulajdonsága. Ennek az értéke lényegében ugyanaz lesz, amelyet a hivatkozás forrásában az `id` tulajdonság értékének megadtunk (nem számít, hogy ez szerepelt-e már a dokumentumban a hivatkozás helye előtt, mert előre és visszafele is lehet hivatkozni).

Az `xref` elem használatakor a hivatkozás szövege magától jön létre, nem tudjuk befolyásolni.

4.50. példa - Az `xref` elem

Tegyük fel, hogy felbukkan a következő szövegrészlet valahol a dokumentumban, amely hivatkozik a korábbi `id` tulajdonságot bemutató példánk azonosítóira:

```
<para>A témával kapcsolatos részleteket az
  <xref linkend="fejezet1"> foglalja össze.</para>

<para>További részleteket pedig a <xref linkend="fejezet1-
szakasz1"> tár
  fel.</para>
```

Ekkor tehát a hivatkozás szövege magától létrejön, így a következő szöveget kapjuk (a *kiemelt rész jelzi* a hivatkozás szövegét):

> A témával kapcsolatos részleteket az *Első fejezet* foglalja össze.
>
> További részleteket pedig az *Első alszakasz* tár fel.

Figyeljük meg hogyan képeződött a fejezet számából vagy a szakasz címéből a megfelelő hivatkozás.

Megjegyzés

Az iméntiekből következik, hogy az `xref` elemmel *nem lehet* `anchor` elemek `id` tulajdonságaira hivatkozni. Az `anchor` elemben nincs semmi, ezért az `xref` nem képes magától létrehozni hozzá a hivatkozás szövegét.

Ha szeretnénk kézzel megadni a hivatkozások szövegét, akkor használjuk a `link` elemet, amelynek a tartalmában szerepeltethetjük ezt.

4.51. példa - A `link` elem

Tegyük fel, hogy a következő szövegrészlet jelenik meg valahol a dokumentumnkban, és az `id` tulajdonságot bemutató példában definiált azonosítókra hivatkozik.

```
<para>Erről bővebb tájékoztatást <link linkend="fejezet1">az ↺
első
  fejezetben</link> kapunk.</para>

<para>Erről a részről pedig <link linkend="fejezet1-
szakasz1">ebben</link> a szakaszban olvashatunk
  többet.</para>
```

Ez a következőképpen jelenik meg (ahol a *kiemelt* szövegek jelzik a hivatkozásokat magukat):

> Erről bővebb tájékoztatást *az első fejezetben* kapunk.
>
> Erről a részről pedig *ebben* a szakaszban olvashatunk többet.

Megjegyzés

Ez utóbbi nem teljesen egy jó példa. Lehetőleg ne „ebben” vagy „itt” néven hivatkozzunk, mert az olvasó így nem fogja közvetlenül látni, hogy az adott hivatkozás pontosan hova is viszi.

Megjegyzés

A `link` elemmel már *tudunk* hivatkozni `anchor` elemek `id` tulajdonságaira, hiszen a `link` elemben már megadható a hivatkozás szövege.

4.2.7.2. A Világhálón található dokumentumok hivatkozása

A külső dokumentumok hivatkozása valamennyivel könnyebb a belső hivatkozások használatánál, mivel ehhez csak annyit kell tudunk, milyen címre akarunk mutatni. Erre

az `ulink` elem alkalmas. Rendelkezik egy `url` tulajdonsággal, amelyben a hivatkozni kívánt oldal címét kell megadnunk. Az elem belsejében pedig a hivatkozás olvasó felé megjelenő szövegét adhatjuk meg.

4.52. példa - Az `ulink` elem

A használat módja:

```
<para>Természetesen már most felhagyhatunk a dokumentum ↺
olvasásával és
  helyette megnézhetjük a <ulink
    url="&url.base;/index.html">FreeBSD honlapját</ulink>.</
para>
```

Így jelenik meg:

Természetesen már most felhagyhatunk a dokumentum olvasásával és helyette megnézhetjük a FreeBSD honlapját.

5. fejezet - * Stíluslapok

Az SGML szabvány nem rendelkezik arról, hogy a dokumentumokat milyen módon kell megjeleníteni a felhasználónak vagy miként kell papírra vetni. Ennek megvalósításához több különböző nyelvet hoztak létre stílusleírások készítéséhez, ilyen például a DynaText, a Panorama, a SPICE, a JSSS, a FOSI, a CSS és a DSSSL.

A stíluslapok DocBook esetében a DSSSL, míg a HTML esetén a CSS szabályai szerint készülnek.

5.1. * DSSSL

A Dokumentációs Projekt Norm Walsh eredeti moduláris DocBook stíluslapjainak némileg módosított változatát használja.

A moduláris stíluslapok a textproc/dsssl-docbook-modular portból érhetőek el.

A Projekt által használt módosított stíluslapok nincsenek a Portgyűjteményben. Ezeket a Dokumentációs Projekt repositoryjában, a `doc/share/xml/freebsd.dsl` állományban találhatjuk meg. A megértését a benne elhelyezett rengeteg megjegyzés segíti, és láthatjuk benne hogyan alakították át a szabványos stíluslapokat a FreeBSD Dokumentációs Projekt igényeinek megfelelően. Ebben az állományban további példákat láthatunk a stíluslap által felismert elemek bővítésére, illetve ezek formázásának leírására. A fejezet elolvasása mellett tehát javasoljuk ennek is az alapos átnézését.

5.2. CSS

A *Cascading Stylesheets (CSS)* egy olyan megoldás, amellyel anélkül tudunk különböző stílusinformációkat (betűtípus, szélesség, méret, szín és így tovább) hozzákapcsolni egy HTML dokumentum elemeihez, hogy erről a dokumentumból bármit is leírnánk.

5.2.1. DocBook dokumentumok

A FreeBSD DSSSL stíluslapok hivatkoznak egy `docbook.css` nevű stíluslapra, amelynek a HTML állományokkal egy könyvtárban kell lennie. A dokumentumok HTML változatának elkészítésekor a `doc/share/misc/docbook.css` állomány fog minden helyre magától bemásolódni és telepítődni.

6. fejezet - A dokumentumok szervezése a doc/ könyvtáron belül

A doc/ könyvtár tartalma egy adott módon szerveződik, és ennek megfelelően a FreeBSD Dokumentációs Projektben készített dokumentumok is adott módon kerülnek elrendezésre. Célunk ezzel megkönnyíteni az újabb dokumentációk felvételét, illetve:

1. leegyszerűsíteni az új dokumentum automatikus átalakítását különböző formátumokba;

2. megteremteni a különböző dokumentumok közti következetes elrendezést, így könnyebben lehet köztük váltani munka közben;

3. segíteni az új dokumentumok helyének egyszerű eldöntésében.

Mindezek mellett a dokumentációt tároló könyvtárnak olyan felépítésűnek kell lennie, amely lehetővé teszi több különböző nyelven és több különböző kódolásban írt dokumentumok kényelmes elhelyezését. Fontos hozzátennünk, hogy a könyvtár szerkezete nem követel meg semmilyen különleges előfeltételezést vagy kulturális berendezkedést.

6.1. A legfelső szint: a doc/ könyvtár

A doc/ további két, különleges névvel és jelentéssel rendelkező könyvtárat rejt.
Könyvtár: share/
Leírás: Ebben a könyvtárban találjuk azokat az állományokat, amelyek függetlenek az egyes fordításoktól és kódolásoktól. A benne található alkönyvtárakon keresztül tovább csoportosítódik a tartalmuk. Például a dokumentáció előállításához kapcsolódó make(1) infrastruktúra állományai a share/mk, miközben a SGML használatához szükséges további állományok (mint például a FreeBSD kiterjesztéseit tartalmazó DocBook DTD) a share/xml alkönyvtárban helyezkednek el.
Könyvtár: nyelv.kódolás/
Leírás: Minden fordításhoz és annak kódolásához tartozik egy könyvtár, például en_US.ISO8859-1 vagy hu_HU.ISO8859-2 . A nevek alapvetően hosszúak, de pontosan meghatározzák az adott nyelvet és a dokumentáció írásához alkalmazott kódolást. Ezzel igyekszük felkészülni olyan esetekre, amikor a fordítócsapatok egy nyelven többféle kódolás szerint is szeretnének dokumentációt készíteni. Ez a megoldás egyben kiutat szolgáltat egy esetleges későbbi, Unicode kódolásra váltás során felmerülő problémák elől.

6.2. A nyelv.kódolás/ könyvtárak

Ezek a könyvtárak tartalmazzák magukat a dokumentumokat. A dokumentumokat ezen a szinten a különböző könyvtárak neveinek megfelelően három vagy több kategóriára osztjuk.
Könyvtár: articles
Tartalom: Az itt található dokumentumokat a DocBook article eleme szerint (vagy egy azzal egyenlő megoldással) jelöltük. Viszonylag rövid, szakaszokra osztott dokumentumokat találhatunk itt. Általában egyetlen HTML állományként érhetőek el.
Könyvtár: books
Tartalom: Ebben a könyvtárban a DocBook book eleme szerint (vagy egy azzal egyenlő megoldással) jelöltük. Hosszabb, fejezetekre osztott dokumentum. Általában egyetlen nagyobb HTML állományként (a gyors internetkapcsolattal rendelkező, vagy a dokumentumot a böngészőből nyomtatni kívánó egyének számára), illetve több kisebb állományként együtteseként is elérhető.
Könyvtár: man
Tartalom: A rendszerhez tartozó man oldalak fordításai. A lefordított szakaszoknak megfelelően ebben a könyvtárban egy vagy több mann nevű alkönyvtárat találhatunk.

Nem mindegyik nyelv.kódolás könyvtár tartalmazza ezeket az alkönyvtárakat. Az egyes fordítások tartalma mindig attól függ, hogy az adott fordítócsapatnak mennyit sikerült eddig lefordítania.

6.3. Az egyes dokumentumokkal kapcsolatos tudnivalók

Ebben a szakaszban a FreeBSD Dokumentációs Projekt keretein belül gondozott különböző dokumentumokat ismerhetjük meg részletesebben.

6.3.1. A kézikönyv

books/handbook/

A kézikönyv a FreeBSD kiterjesztéseit tartalmazó DocBook DTD szerint készült.

A kézikönyv a DocBook book elemének megfelelően szerveződik. Több part elemmel jelölt részből áll, amelyek mindegyike több chapter elemmel jelölt fejezetet foglal magában. Ezek a fejezetek további szakaszokra (sect1) bomlanak, amelyek helyenként alszakaszokra (sect2, sect3) oszlanak, és így tovább.

6.3.1.1. Fizikai szervezés

A kézikönyv forrásai több különböző állományban és könyvtárban a handbook könyvtáron belül találhatóak.

Megjegyzés

A kézikönyv szervezése időről-időre változik, ezért könnyen előfordulhat, hogy ez a dokumentum csak kissé késve követi ezeket a változtatásokat. Ha további kérdéseink lennének a kézikönyv szervezéséről, bátran írjunk a FreeBSD Dokumentációs Projekt levelezési lista címére!

6.3.1.1.1. Makefile

A Makefile állományban definiálódnak olyan változók, amelyek a SGML források különböző formátumúra alakításának menetét befolyásolják, illetve hivatkozik a kézikönyv forrásaira. Ezután beemeli a doc.project.mk állományt, és így lényegében betölti a dokumentumok átalakításáért felelős további utasításokat.

6.3.1.1.2. book.xml

Ez a kézikönyv legfelső szintű dokumentuma. Ebben van a kézikönyv dokumentípus-deklarációja, illetve a szerkezetét leíró további elemek.

A book.xml az .ent kiterjesztésű állományokat paraméteregyedek segítségével tölti be. Ezek az állományok (amelyekről később még szó lesz) aztán a kézikönyv további részeiben használt általános egyedeket definiálnak.

6.3.1.1.3. könyvtár/chapter.xml

A kézikönyv egyes fejezetei egymástól különálló könyvtárakban, chapter.xml nevű állományokban tárolódnak. Ezeket a könyvtárakat az adott fejezetet jelölő chapter id tulajdonsága szerint szokták elnevezni.

Például ha az egyik fejezet forrásában a következő sor olvasható:

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

Ekkor a chapter.xml nevű állományt tartalmazó könyvtár neve kernelconfig lesz. Egy ilyen állomány általában a teljes fejezetet tartalmazza.

A kézikönyv HTML változatának készítése során ebből a kernelconfig.html állomány fog keletkezni. Ezt azonban az id értéke határozza meg, semmi köze nincs a könyvtár elnevezéséhez.

A kézikönyv korábbi változataiban az összes forrás a `book.xml` állománnyal volt egy szinten, és az adott `chapter` elemek `id` tulajdonságának megfelelően került elnevezésre. Az egyes fejezetekhez most már külön-külön tudunk képeket csatolni, amelyeket a fejezeteknek megfelelő könyvtárban kell elhelyezni a `share/images/books/handbook` könyvtáron belül. Ha honosítani akarjuk a képeket, akkor viszont ügyeljünk arra, hogy az adott fejezet könyvtárába, az SGML források mellé tegyük a lefordított képeket. A névütközés egy idő után úgyis elkerülhetetlenné válik, és sok, kevés állományt tartalmazó könyvtárral egyébként is könnyebb dolgozni, mint egy sok állományt tartalmazó könyvtárral.

A kézikönyv forrásaiban könnyen láthatjuk, hogy sok ilyen könyvtár van, bennük egy-egy `chapter.xml` állománnyal. Például `basics/chapter.xml` , `introduction/chapter.xml` és `printing/chapter.xml` .

Fontos

A fejezeteket és könyvtárakat nem szabad semmilyen sorrendiségre utaló módon elnevezni. A fejezetek elrendezése ugyanis változhat a kézikönyv egy esetleges átszervezése során. Az ilyen átszervezések során pedig (általában) nem lenne szabad állományokat átnevezni (hacsak komplett fejezeteket nem mozgatunk fentebb vagy lentebb a szerkezetben).

Az egyes `chapter.xml` állományok önmagukban teljes SGML dokumentumok. Különösen azért, mert semmilyen `DOCTYPE` sor nem található az elejükön.

Ez abból a szempontból hátrányos, hogy ezeket az állományokat ezért nem tudjuk normál SGML állományokként kezelni. Emiatt ezeket nem lehet egyszerűen, a kézikönyvhöz hasonlóan módon HTML, RTF, PS vagy más egyéb formátumba átalakítani. Ezért tehát könnyen előfordulhat, hogy a fejezetek megváltoztatásakor a *teljes* kézikönyvet elő kell állítanunk.

7. fejezet - A dokumentáció előállításának folyamata

Ebben a fejezetben szeretnénk pontosan tisztázni *hogyan szerveződik a dokumentáció előállításának folyamata* és *hogyan tudunk ebbe beavatkozni.*

A fejezet elolvasása során megismerjük:

- az SGML eszközeiről szóló fejezetben említetteken túl a FreeBSD Dokumentációs Projekt keretein belül készített dokumentáció különböző változatainak előállításához mire van még szükségünk;
- a dokumentumokhoz tartozó `Makefile` állományokban szereplő `make` utasításokat, valamint a hivatkozott `doc.project.mk` vázlatos felépítését;
- további `make` változókon és célokon keresztül miként tudjuk testreszabni a dokumentáció különböző változatainak előállítási folyamatát.

7.1. A FreeBSD dokumentáció előállításának eszközei

Munkánk folyamán az itt felsorolt eszközök állnak rendelkezésünkre. Használjuk ki az általuk nyújtott lehetőségeket, amennyire csak tudjuk.

- Az elsődleges eszköz maga a `make` parancs, pontosabban a Berkeley Make.
- Csomagokat a FreeBSD alaprendszerében megtalálható `pkg_create` programmal tudunk készíteni. Ha nem FreeBSD alatt dolgozunk, akkor vagy csomagok nélkül kell dolgoznunk, vagy magunknak kell ezeket elkészítenünk.
- A `gzip` segítségével lehet az előállított dokumentumok tömörített változatát elkészíteni. Emellett még a `bzip2` és `zip` típusú tömörítés is támogatott. A `tar` programot is támogatjuk, a csomagok készítéséhez kell.
- A dokumentáció telepítésének elfogadott eszköze az `install` program. Természetesen léteznek egyéb megoldások is.

Megjegyzés

Nem valószínű, hogy ez az utolsó két eszközt ne lenne elérhető a rendszerünkön, csupán a teljesség kedvéért említettük meg ezeket.

7.2. A dokumentációt tároló könyvtárban található Makefile állományok

A FreeBSD Dokumentációs Projekt által használt könyvtárakban megtalálható Makefile állományoknak három típusa létezik:

- Az alkönyvtári Makefile állományok egyszerűen csak továbbadják a parancsokat az alkönyvtáraiknak.
- A dokumentumokra vonatkozó Makefile állományok írják le, hogy milyen dokumentumokat kellene az adott könyvtárban előállítani.
- Az .mk állományok segítik valamilyen formában a dokumentumok előállítását. Többnyire doc.xxx.mk névvel láthatóak.

7.2.1. Az alkönyvtári Makefile állományok

Ezek a típusú Makefile állományok általában a következő alakúak:

```
SUBDIR =articles
SUBDIR+=books

COMPAT_SYMLINK = en

DOC_PREFIX?= ${.CURDIR}/..
.include -"${DOC_PREFIX}/share/mk/doc.project.mk"
```

Röviden összefoglalva: az első négy nem üres sorban ún. make változókat definiálunk. Ezek rendre a SUBDIR, COMPAT_SYMLINK és DOC_PREFIX.

Az első SUBDIR sornál, illetve a COMPAT_SYMLINK sorában láthatjuk hogyan kell egy új értéket beállítani egy ilyen változónak.

A második SUBDIR sorban azt láthatjuk, hogyan tudunk a változó aktuális értékéhez továbbiakat hozzáfűzni. Ebben az esetben tehát az utasítás végrehajtása után a SUBDIR értéke articles books lesz.

A DOC_PREFIX esetében pedig olyan értékadást figyelhetünk meg, amelyik csak akkor hajtódik végre ténylegesen, ha a változónak addig még nem volt értéke. Ez olyankor

tud kapóra jönni, amikor a DOC_PREFIX nem pontosan az, amire a Makefile számít — a felhasználó ekkor meg tudja adni a helyes értéket.

Ez így együttesen tehát mit is jelent? A SUBDIR összefoglalja azokat a könyvtárakat, amelyekben a dokumentumok előállításának folyamatának folytatódnia kell majd.

A COMPAT_SYMLINK a kompatibilitás céljából létrehozott szimbolikus linkekre vonatkozik, amelyek (valamilyen csoda folytán) az adott nyelv hivatalos kódolására mutatnak (tehát például a doc/en a en_US.ISO8859-1 könyvtárra).

A DOC_PREFIX a FreeBSD Dokumentációs Projekt főkönyvtárához vezető utat adja meg. Ezt nem mindig egyszerű megtalálni, ezért a rugalmasság kedvéért könnyedén felül is definiálható. A .CURDIR a make egyik saját belső változója, amelyben az aktuális könyvtár elérési útját tárolja.

Végül az utolsó sorban a FreeBSD Dokumentációs Projekt összes Makefile állományára vonatkozó, rendszerszintű doc.project.mk állományra hivatkozunk, amelyen keresztül az iménti változókból épül fel a dokumentumok előállításának pontos menete.

7.2.2. A dokumentumokra vonatkozó Makefile állományok

Ezekben a Makefile állományokban az adott könyvtárban található dokumentumok előállítását leíró különböző make változók szerepelnek.

Lássunk erre egy példát:

```
MAINTAINER=pgj@FreeBSD.org

DOC?= book

FORMATS?= html-split html

INSTALL_COMPRESSED?= gz
INSTALL_ONLY_COMPRESSED?=

# Az SGML forrás
SRCS=  book.xml

DOC_PREFIX?= ${.CURDIR}/../../..

.include -"$(DOC_PREFIX)/share/mk/docproj.docbook.mk"
```

A MAINTAINER változó nagyon fontos. A FreeBSD Dokumentációs Projekten belül ezen a változón keresztül jelezhetjük a dokumentum birtoklását, vagyis karbantartási kötelezettségünket.

A DOC hivatkozik (az .xml kiterjesztés nélkül) az adott könyvtárban található dokumentum fő forrására. Emellett az SRCS változóban kell összefoglalnunk a

dokumentumot alkotó források neveit. Ebben érdemes megadni minden olyan állományt, amelynek megváltozása esetén újra elő kell állítani az érintett dokumentumot.

A `FORMATS` segítségével definiáljuk a dokumentum alapértelmezetten előállítandó formátumait. A `INSTALL_COMPRESSED` változóban a dokumentum elkészítésekor felhasználandó tömörítési formákat adhatjuk meg. A `INSTALL_ONLY_COMPRESSED` változó alapból üres, de ha adunk neki valamilyen egyéb értéket, akkor a dokumentumoknak csak a tömörített változata fog elkészülni.

Megjegyzés

A változók feltételes értékadásáról már volt szó az előző szakaszban.

A `DOC_PREFIX` változó és az `.include` utasítás a korábbiak alapján már ismerős lehet.

7.3. A FreeBSD Dokumentációs Projekt `.mk` állományai

Ezek az állományok legjobban talán önmagukon keresztül mutathatóak be. A következő rendszerszintű `.mk` állományokat használjuk a FreeBSD Dokumentációs Projektben:

- A `doc.project.mk` a központi `.mk` állomány, amely szükség szerint hivatkozik az összes többi `.mk` állományra.
- Az előállítás és a telepítés során a `doc.subdir.mk` felelős a dokumentumokat tároló könyvtárak bejárásért.
- A `doc.install.mk` tartalmazza a karbantartóval és a telepítéssel kapcsolatos változókat.
- A `doc.docbook.mk` állomány csak akkor kerül feldolgozásra, ha a `DOCFORMAT` értéke `docbook` és a `DOC` változónak van értéke.

7.3.1. A `doc.project.mk` állomány

Nézzünk bele:

```
DOCFORMAT?= docbook
MAINTAINER?= doc@FreeBSD.org

PREFIX?= /usr/local
PRI_LANG?= en_US.ISO8859-1

.if defined(DOC)
```

```
.if ${DOCFORMAT} == -"docbook"
.include -"doc.docbook.mk"
.endif
.endif

.include -"doc.subdir.mk"
.include -"doc.install.mk"
```

7.3.1.1. Változók

Ha nem állítjuk be a dokumentum `Makefile` állományában, akkor a `DOCFORMAT` és a `MAINTAINER` változók ezen a helyen kapnak értéket.

A `PREFIX` adja azt a könyvtárat, amelyen belül elérhetőek a dokumentáció előállításához szükséges eszközök. A csomagok és portok átlagos használata esetén ez a `/usr/local` .

A `PRI_LANG` adja meg azt a nyelvet és kódolást, amely a dokumentációt olvasó felhasználó számára elsődlegesként leginkább elfogadott. Alapértelmezés szerint ez az amerikai angol.

Megjegyzés

A `PRI_LANG` változó semmilyen hatással nincs a dokumentumok előállítására. Egyedül a FreeBSD dokumentáció telepítésekor a leggyakrabban hivatkozott dokumentumokhoz létrehozandó szimbolikus linkek készítésénel van szerepe.

7.3.1.2. Elágazások

A `.if defined(DOC)` sorban a `Makefile` állományokban megadható elágazásokra láthatunk példát. Hasonlóan más programokhoz, a `Makefile` működését tudjuk meghatározni egy logikai kifejezés igazságértéktől függően. Ebben a kifejezésben a `defined` függvény, amely megadja, hogy a paramétereként megadott változó definiált-e.

A következő elágazásban, vagyis az `.if ${DOCFORMAT} == "docbook"` utasításban azt vizsgáljuk meg, hogy a `DOCFORMAT` változó értéke `"docbook"` vagy sem. Amennyiben a válasz erre igen (vagyis „igaz”), beemeljük a `doc.docbook.mk` tartalmát.

Az előbb említett két elágazást rendre az `.endif` kulcsszóval zárjuk le.

7.3.2. A `doc.subdir.mk` állomány

Ez az állomány már túlságosan nagy ahhoz, hogy a fejezeten belül könnyen ki lehessen elemezni. Ezért az előző szakaszok alapján a részleteket a kedves Olvasóra bízzuk, ehhez adunk még itt némi segítséget.

7.3.2.1. Változók

- A SUBDIR tartalmazza azokat az alkönyvtárakat, amelyeket a feldolgozás során be kell járnunk.
- A ROOT_SYMLINKS a dokumentáció főkönyvtárából szimbolikusan linkelendő könyvtárak neveit adja meg, amennyiben az adott nyelv (a PRI_LANG változó szerint) az elsődleges.
- A COMPAT_SYMLINK változót már korábban bemutattuk az alkönyvtári Makefile állományok című szakaszban.

7.3.2.2. Célok és makrók

A függőségi viszonyokat cél: függőség1 függőség2 ... formában írjuk fel, ahol így megmondjuk, hogy a cél létrehozásához először milyen elemeknek kell léteznie. Ezeket nevezzük függőségeknek.

A függőségi viszony megadása alatt lehetőségünk van részletezni a függőségekből a cél előállításához szükséges utasításokat. Ezt akkor kell megtenni, ha a cél és a függőségek közti átalakítást előzőleg még nem definiáltuk, vagy ha az adott esetben az átalakítás eltér a korábbiaktól.

A .USE nevű speciális függőség egy makróval egyenértékű eszköz használatára ad lehetőséget.

```
_SUBDIRUSE: -.USE
.for entry in ${SUBDIR}
 @${ECHO} -"===> ${DIRPRFX}${entry}"
 @(cd ${.CURDIR}/${entry} && \
 ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} ↺
DIRPRFX=${DIRPRFX}${entry}/ -)
.endfor
```

A fenti kódrészletben tehát a _SUBDIRUSE most már egy „makró” lesz, amely ha megjelenik a függőségek között, akkor a törzsében megadott parancsokat hajtja végre.

Mi különbözteti meg ezt a makrót a többi céltól? Két lényeges eltérés: először is, a benne megadott utasítások a rá függőségként hivatkozó célhoz társított átalakítást végző utasítások *után* fognak lefutni, másrészt nem befolyásolja a jelenleg feldolgozás alatt álló cél nevét tároló .TARGET változó értékét.

```
clean: _SUBDIRUSE
 rm --f ${CLEANFILES}
```

Ebben a kódrészletben a tehát clean esetében csak az rm -r ${CLEANFILES} parancs lefutása után fog végrehajtódni a _SUBDIRUSE makró tartalma. Ennek hatására a clean megy egyre lentebb és lentebb a könyvtárszerkezetben, *miközben* törli a előzőleg előállított állományokat.

7.3.2.2.1. Definiált célok

- Az `install` és a `package` célok egyaránt folyamatosan haladnak lefelé a könyvtárszerkezetben és az alkönyvtárakban hívják saját maguk tényleges változatát (ezek a `realinstall` és `realpackage`).

- A `clean` eltávolítja a folyamat során keletkezett állományokat (és az előbbiekhez hasonlóan lefele halad a könyvtárszerkezetben). A `cleandir` ugyanezt csinálja, de ha talál a tárgykódokhoz tartozó könyvtárat, akkor azt is törli.

7.3.2.3. Bővebben a feltételes kifejezésekről

- Az `exists` egy másik logikai függvény, amellyel lekérdezhetjük, hogy a paramétereként megadott állomány létezik-e.

- Az `empty` logikai függvény igaz értékű, ha a paramétereként megadott változó értéke üres.

- A `target` logikai függvény igaz értékű, ha a paraméterként megadott cél még nem létezik.

7.3.2.4. Ciklusszerverzési lehetőségek (`.for`)

A `.for` utasítás segítségével adott utasításokat tudunk elvégezni egy változó tartalmaként megadott, szóközökkel határolt elemekre. A ciklus belsejében egy változóból érhetjük el az aktuálisan feldolgozott elemet.

```
_SUBDIRUSE: -.USE
.for entry in ${SUBDIR}
 @${ECHO} -"===> ${DIRPRFX}${entry}"
 @(cd ${.CURDIR}/${entry} && \
 ${MAKE} ${.TARGET:S/realpackage/package/:S/realinstall/install/} ↺
DIRPRFX=${DIRPRFX}${entry}/ -)
.endfor
```

A fenti kódrészletben ha a `SUBDIR` üres, akkor nem történik semmi. Ha viszont egy vagy több elemet is tartalmaz, akkor a `.for` és az `.endfor` között megadott utasítások megismétlődnek minden egyes elem esetén. Ezek értékét a ciklus belsejében rendre a `entry` változóban veszi fel.

8. fejezet - A honlap

8.1. Előkészületek

A honlap előállításához elsősorban elegendő szabad területet kell keresnünk valamelyik merevlemezünkön. Ennek mennyisége a választott módszertől függően úgy nagyjából 200 MB-tól 500 MB-ig terjedhet. Ez a becslés magában foglalja az SGML eszközökhöz, a CVS repository megfelelő részeihez, valamint a honlap generálásához szükséges lemezterületet.

Megjegyzés

Mindig ellenőrizzük, hogy a dokumentáció előállításához használt portok frissek legyenek. Ha nem vagyunk benne biztosak, akkor a portok telepítése előtt a pkg_delete(1) paranccsal töröljük a korábbi változatukat. Például ha a jade-1.2 csomagra van szükségünk és a rendszerünkön már megtalálható a jade-1.1, akkor a következőt kell tennünk:

```
# pkg_delete jade-1.1
```

A honlap előállításához ebben a fejezetben most két módszert adunk meg:

- A `csup` parancs használatával hozzuk létre és tartsunk karban a gépünkön a források helyi másolatát valamelyik CVSup szerverről. Ez a legegyszerűbb megoldás, mivel semmilyen további szoftver telepítését nem igényli. Ehhez a következő szakaszban megadott supfile állomány mindig a szükséges állományok legfrissebb változatát kéri le. Ez abban az esetben tökéletesen megfelelő, ha egyszerűen csak le akarjuk generálni a honlapokat és nem kívánunk a forrásokkal dolgozni.

Megjegyzés

A csup(1) a FreeBSD 6.2-RELEASE kiadástól az alaprendszer része. Amennyiben még a FreeBSD egy korábbi változatát használjuk, akkor ezt a programot a Portgyűjteményből a net/csup port telepítésével érhetjük el.

- A cvsup parancs használatával „CVS módban” hozzunk létre és tartsunk karban egy helyi CVS repositoryt a szükséges állományokkal. Ehhez például a net/cvsup-without-gui port telepítését kell elvégeznünk, ezáltal viszont egy sokkal rugalmasabb módszert nyerünk abban az esetben, ha gyorsan és könnyedén hozzá szeretnénk férni a doc és www repositorykban tárolt állományok különböző revízióihoz, előzményeihez vagy éppen tárolni szeretnénk a FreeBSD központi CVS repositoryjába.

8.1.1. Az egyszerű megoldás: a csup használata

A csup az alaprendszer része lett, de egy ideje már nagyon sokan használják, többek közt a saját portgyűjteményük frissítésére. A most következő példa supfile állománnyal a honlapok előállításához szükséges állományokat tudjuk elérni:

```
#
# Ezzel a konfigurációs állománnyal a FreeBSD honlapjának
# legenerálásához szükséges gyűjteményeket tudjuk elérni.
#
# A http://www.freebsd.org/doc/handbook/mirrors.html honlapon ↺
található
# felsorolásból válasszuk ki a hozzánk legközelebb elhelyezkedő CVSup
# tükrözést.

*default host=cvsup10.FreeBSD.org
*default base=/var/db
*default prefix=/usr/build
*default release=cvs tag=.
*default delete use-rel-suffix
*default compress

# A FreeBSD repository teljes doc ágát lekérjük.

doc-all

# Lekérjük a honlap forrásait.

www

# A holnapok előállításához szükséges néhány alapvető port lekérése.

ports-base
```

A default host helyére természetesen ne felejtsük el megadni a hozzánk legközelebb elhelyezkedő CVSup tükrözést (például cvsup.hu.FreeBSD.org), illetve a default prefix bejegyzésnél azt a könyvtárat, ahová a lekért állományokat szeretnénk elhelyezni. Ezután az így kitöltött mintát mentsük el például doc-www-supfile néven és adjuk ki a következő parancsot:

```
# csup --g --L2 doc-www-supfile
```

A parancs lefutásának eredményeképpen ekkor tehát a `default prefix` értékeként megadott könyvtáron belül létrejönnek a `doc/`, `www/` és `ports/` alkönyvtárak (amely a példánkban a `/usr/build` volt). Ebben a könyvtárban fogjuk egyébként létrehozni az állományokat, ezért ezt érdemes egy olyan állományrendszerre tenni, ahol tehát elegendő szabad terület áll rendelkezésre.

Remek! Most lépjünk tovább a honlap előállításáról szóló részhez.

8.1.2. A rugalmasabb megoldás: saját doc és www repositoryk létrehozása és karbantartása

Ez a módszer több lehetőséget kínál, viszont cserébe telepítenünk kell hozzá a net/cvsup-without-gui portot vagy csomagot.

Megjegyzés

A net/cvsup-without-gui port fordításához szükséges a lang/ezm3 port, vagyis egy Modula 3 fordító. Ennek a fordítása viszonylag sok időt vesz igénybe és ráadásul a legtöbben nem is használják másra, ezért a CVSup telepítéséhez elsősorban a csomagok használatát javasoljuk.

A CVSup program rendelkezik egy speciális, ún. „CVS móddal”, amelynek köszönhetően lehetővé teszi a CVS repositoryt alkotó `,v` állományok elérését. Ez a funkció jelenleg még nem érhető el a csup programban. A CVSup részletes bemutatását a FreeBSD kézikönyv A források szinkronizálása című részében olvashatjuk.

A most következő supfile állomány lekéri a holnapok előállításához szükséges gyűjteményeket és létrehozza a CVS repository helyi másolatát:

```
#
# Ezzel az állománnyal létre tudjuk hozni a CVS repository egy olyan
# helyi másolatát, amelyben csak a FreeBSD holnapjának előállításához
# szükséges gyűjtemények találhatóak meg.  Jelen pillanatban ↺
*kizárólag*
# a cvsup paranccsal fog működni (a csup programmal tehát nem)

*default host=cvsup10.FreeBSD.org
*default base=/var/db
*default prefix=/usr/dcvs
*default release=cvs
*default delete use-rel-suffix
*default compress

# A honlapok generálásához az alábbi gyűjteményekre lesz szükségünk:
```

```
ports-base
doc-all
www

# A CVS funkciókhoz még ezek a gyűjtemények is kelleni fognak:

cvsroot-common
cvsroot-ports
cvsroot-doc
```

A `default host` sorban értelemszerűen a hozzánk legközelebb elhelyezkedő CVSup tükrözést adjuk meg (például `cvsup.hu.FreeBSD.org`), illetve a `default prefix` bejegyzésnél pedig azt a könyvtárat, ahol a repository állományait kívánjuk tárolni. Valamilyen, például `doc-ww-cvsfile` néven mentsük el ezt a mintát és adjuk ki a következő parancsot:

```
# cvsup --g --L2 doc-www-cvsfile
```

Továbbá érdemes még a parancsértelmezőnk indítóállományaiban beállítani a `CVSROOT` környezeti változó értékét is. Például vegyük fel az alábbi sort a `.cshrc` állományunkba:

```
setenv CVSROOT /usr/dcvs
```

Ennek megadásával a repositoryval kapcsolatos műveletek elvégzésekor a (lentebb látható) parancsból elhagyhatjuk a `-d` paraméter megadását.

Jelenleg a repositoryban tárolt állományok befogadásához legalább 400 MB tárterületre lesz szükségünk. A honlapok előállítása során ezen felül ideiglenesen még nagyjából további 200 MB hely kellhet. A `cvsup` parancs lefutása után már ki is kérhetjük a forrásokat a munkakönyvtárunkba:

```
# mkdir -/usr/build
# cd -/usr/build
# cvs --d -/usr/dcvs --R co --AP doc www ports
```

Ez a parancs lényegében ugyanannak felel meg, ahogy a csup kéri le az állományokat a CVSup szervertől. A folyamat befejeződése után a munkakönyvtárban tehát tulajdonképpen ugyanazokat fogjuk találni, mint az egyszerűbb, csup alapú módszer esetében.

Az imént bemutatott `cvsup` parancs folyamatos használatával tudjuk rendszeresen karbantartani a CVS repository helyi másolatát. Az első esetben még viszonylag sok állomány fog letöltődni, azonban a későbbiekben már viszont csak néhány percet vesz igénybe a frissítés.

8.2. A honlapok előállítása

Miután az előbb tárgyalt módszerek valamelyikével előkészítettük rendszerünkön a honlapok forrásainak egy naprakész másolatát, készen állunk a honlapok létrehozására. A példánkban az ehhez használt munkakönyvtár a `/usr/build` volt, ahol már minden szükséges állomány megtalálható.

1. Lépjünk be a munkakönyvtárba:

```
# cd -/usr/build
```

2. A honlapok előállítása a `www/en` könyvtárból indul, az `all` make(1) cél végrehajtásával megkezdődik a honlapok készítése.

```
# cd www/en
# make all
```

8.3. A generált honlapok telepítése a webszerverre

1. Ha nem az en könyvtárban állunk, akkor váltsunk vissza rá.

```
# cd -/usr/build/www/en
```

2. A `DESTDIR` változóban állítsuk be a honlapok tényleges helyét, és futtassuk le vele a `install` make(1) célt.

```
# env DESTDIR=/usr/local/www make install
```

3. Ha az előbb megadott könyvtárba korábban már másoltunk honlapokat, akkor az újabb másolás során nem törlődnek a régi vagy elavult lapok. Például ha a honlapokat napi rendszereséggel frissítjük, akkor a következő paranccsal meg tudjuk keresni és törölhetjük azokat a lapokat, amelyeket már három napja nem frissítettünk.

```
# find -/usr/local/www --ctime 3 --print0 -| xargs --0 rm
```

8.4. Környezeti változók

`CVSROOT`
: A CVS állományait tároló könyvtár gyökere. Ha a CVSup alapú módszert alkalmazzuk, akkor érdemes a hozzá tartozó változót beállítanunk:

```
# CVSROOT=/usr/dcvs; export CVSROOT
```

A `CVSROOT` egy környezeti változó. Vagy a paranccsorban, vagy pedig a parancsértelmezőnknek megfelelő konfigurációs állományban (például `.profile`) kell beállítanunk. Ennek pontos mikéntjét maga a parancsértelmező határozza meg

(a fenti parancsban például a bash és a hozzá hasonló parancsértelmezők által alkalmazott megadási mód látható).

ENGLISH_ONLY
: Ha beállítjuk és nem üres, akkor a folyamat során csak az angol nyelvű oldalak fognak elkészülni, a fordítások figyelmen kívül maradnak. Például:

```
# make ENGLISH_ONLY=YES all install
```

Ha le akarjuk tiltani az ENGLISH_ONLY hatását és az összes oldalt az összes elérhető fordítással létrehozni, akkor az ENGLISH_ONLY változónak adjunk üres értéket:

```
# make ENGLISH_ONLY="" all install clean
```

WEB_ONLY
: Ha beállítottuk és az értéke nem üres, akkor hatására csak a www könyvtárban található HTML oldalak állítódnak elő és telepítődnek. Ilyenkor a doc könyvtár teljes tartalma (kézikönyv, GYIK és egyéb leírások) figyelmen kívül marad.

```
# make WEB_ONLY=YES all install
```

WEB_LANG
: Ha beállítottuk, akkor a www könyvtáron belül csak a benne megadott nyelvekhez tartozó könyvtárak fognak előállítódni. Az angol kivétel tehát ilyenkor minden más nyelv kimarad a feldolgozásból. Például:

```
# make WEB_LANG="el es hu nl" all install
```

NOPORTSCVS
: Ennek megadásakor a Makefile állományok nem kérnek ki állományokat a portokhoz tartozó repositoryból. Ehelyett a szükséges állományokat közvetlenül a /usr/ports könyvtárból (vagy ahova a PORTSBASE változó értéke mutat) fogják átmásolni.

A WEB_ONLY, WEB_LANG, ENGLISH_ONLY és NOPORTSCVS változók a make programhoz tartoznak. Ezek értékét az /etc/make.conf állományban, vagy környezeti változókhoz hasonlóan parancssorból, illetve a parancsértelmező konfigurációs állományaiban állíthatjuk be.

9. fejezet - Fordítások

Ebben a fejezetben gyakran ismételt kérdések és válaszok formájában próbálunk segítséget nyújtani a FreeBSD dokumentációjának (a kézikönyv, a GYIK, különböző leírások, man oldalak és egyebek) különböző nyelvekre fordításában.

Az itt szereplő bejegyzések *nagyrészt* az eredetileg Frank Gründer (<elwood@mc5sys.in-berlin.de >) által a Német FreeBSD Dokumentációs Projekt számára összeállított GYIK tartalmán alapszanak, amelyet Bernd Warken (<bwarken@mayn.de >) fordított később angolra.

A bejegyzéseket jelenleg a Documentation Engineering Team <doceng@FreeBSD.org > tartja karban.

K: Miért éppen GYIK?

V: A freebsd-doc levelezési listán egyre többen és többen jelzik, hogy szeretnék lefordítani a FreeBSD dokumentációját különböző idegennyelvekre. Az itt összegyűjtött kérdésekben ezért most igyekszünk megválaszolni az ilyenkor általában előkerülő problémákat, hogy minél gyorsabban el tudják kezdeni a munkát.

K: Mit az az i18n és l10n?

V: Az i18n jelentése internationalization (idegennyelvűség), az l10n jelentése pedig localization (honosítás). Ezeket egyszerűsítették le és rövidítették.

Az i18n úgy értelmezhető, hogy először egy „i”, majd 18 betű, aztán egy „n”. Ehhez hasonlóan, az l10n egy „l”, amelyet 10 betű követ és egy „n” zár.

K: Van külön levelezési lista a fordítók számára is?

V: Igen. Az egyes fordítói csapatoknak többnyire van saját önálló levelezési listájuk. Ezzel, illetve a csapatok által működtetett webhelyekkel kapcsolatban a fordítói projektekkel foglalkozó oldalon találhatunk bővebb információkat.

K: Szükség van még fordítókra?

V: Igen. Minél többen dolgoznak egy fordításon, annál gyorsabban készül el, illetve annál gyorsabban frissül az eredeti angol dokumentáció változásai szerint.

Nem kell képzett szakfordítónak lenni ahhoz, hogy segíteni tudjunk.

K: Milyen nyelveket kell ismerni?

V: Nem árt jól ismernünk az írott angolt és értelemszerűen folyékonyan beszélni a fordítás célnyelvét.

Az angol nyelv ismerete egyébként nem kötelező. Például a GYIK spanyol fordítását elkészíthetjük a magyar változat alapján is.

K: Milyen szoftvereket kell ismerni?

V: Mindenképpen javasoljuk, hogy hozzunk létre magunknak egy helyi másolatot a FreeBSD repositoryjáról (legalább a dokumentációról) a CTM vagy a CVSup segítségével. Az említett alkalmazások használatáról a FreeBSD kézikönyv A forrás szinkronizálása című szakaszában olvashatunk részletesebben.

Hasznos, ha járatosak vagyunk a CVS használatában. Segítségével meg tudjuk nézni, hogy mi változott a különböző revíziókban, így mindig csak a változások lefordításával karban tudjuk tartani a lefordított dokumentációkat.

K: Honnan lehet kideríteni, hogy esetleg valaki más már dolgozik ugyanazon nyelv fordításán?

V: A Dokumentációs Projekt fordításokkal foglalkozó oldalán megtalálhatjuk a jelenleg ismert összes fordítást. Ha valaki vagy valakik már dolgoznak fordításon a kiválasztott nyelvhez, akkor inkább vegyük fel velük a kapcsolatot, hogy ne dolgozzon senki sem feleslegesen.

Attól függetlenül, hogy az említett oldal szerint senki sem foglalkozik az adott nyelvre fordítással, a biztonság kedvéért küldjünk még egy levelet a FreeBSD Dokumentációs Projekt levelezési lista címére. Előfordulhat ugyanis, hogy hozzánk hasonlóan valaki szintén szeretne fordítani, de hivatalosan még nem jelentette be.

K: Senki sem fordít a kiválasztott nyelvre. Mi a teendő?

V: Nos, ebben az esetben gratulálunk, miénk a „*Nyelv* FreeBSD Dokumentációs Projekt”! Isten hozott a fedélzeten!

Elsőként alaposan fontoljuk meg, hogy valóban hajlandóak vagyunk kellő időt szentelni rá. Mivel jelen pillanatban egyedül csak mi foglalkozunk az adott nyelvi fordítással, nekünk magunknak kell képviselnünk és népszerűsítenünk a munkánkat, illetve irányítani a később csatlakozni kívánó önkéntesek munkáját.

Írjunk egy levelet a FreeBSD Dokumentációs Projekt levelezési listájára, amelyben bejelentjük, hogy nekikezdünk fordítani az adott nyelvre, ezáltal felkerül az előbb említett honlapra.

Ha az adott nyelvhez tartozó országban van a FreeBSD Projektnek valamilyen tükrözése, akkor érdemes kapcsolatba lépnünk az üzemeltetőitől és kérni a munkánkhoz némi tárhelyet, esetleg a levelezés támogatását vagy saját egy levelezési listát.

Válasszunk egy dokumentumot és kezdjük el fordítani. Érdemes valamelyik rövidebb résszel kezdeni, például a GYIK-kal vagy valamelyik leírással.

K: Hova lehet küldeni fordításokat?

V: Ez változó. Ha már az adott nyelven dolgozik egy fordítócsapat (például a magyar vagy a német), akkor a saját honlapjukon valószínűleg megadják hogyan kezelik és hogy lehet hozzájuk eljuttatni a fordításokat.

Amennyiben viszont még csak egyedül dolgozunk az érintett nyelven (vagy a fordítócsapatunk képviseletében szeretnénk eljuttatni a munkánkat a FreeBSD Projektnek), akkor érdemes közvetlenül a FreeBSD Projektnek küldeni a fordításainkat (lásd a következő kérdést).

K: Hova lehet beküldeni a fordításokat, ha senki más nem dolgozik még fordításon az adott nyelvhez?

avagy:

A fordítócsapatok hova tudják küldeni a tagjaik által készített fordításokat?

V: Először is az elkészült fordításokat a megfelelő formára kell hoznunk. Ez nagyjából azt jelenti, hogy tegyük a már meglevő dokumentációk közé és próbáljuk meg előállítani belőle a különböző formátumokat.

A FreeBSD dokumentációja jelenleg a legfelső szinten egy `doc/` könyvtárba szerveződik. A benne található könyvtárakat az adott nyelvek ISO639 szabványú (a FreeBSD 1999. január 20. utáni változataiban a `/usr/share/misc/iso639` állományban definiált) kódja szerint nevezik el.

Ha az adott nyelv többféle kódolással is rendelkezik (mint például a kínai), akkor ezen a szinten az egyes kódolásokhoz külön könyvtárak fognak tartozni.

Végezetül az egyes dokumentumokat tegyünk külön könyvtárakba.

Például egy képzeletbeli svéd fordítás körülbelül így nézne ki:

```
doc/
    sv_SE.ISO8859-1/
                      Makefile
                      books/
                            faq/
                                Makefile
                                book.xml
```

Az `sv_SE.ISO8859-1` a fordítás neve, amely tehát a korábban tárgyalt `nyelv.kódolás` alakban szerepel. A dokumentáció előállításához elhelyeztünk még benne két `Makefile` állományt is.

A tar(1) és gzip(1) programok segítségével tömörítsük össze az így összekészített dokumentációt és juttassuk el a Projekthez.

```
% cd doc
```

```
% tar cf swedish-docs.tar sv_SE.ISO8859-1
% gzip --9 swedish-docs.tar
```

Az így keletkező `swedish-docs.tar.gz` állományt töltsük fel valahova. Ha nincs saját tárhelyünk az interneten (mert például a szolgáltatónk nem bocsátott a rendelkezésünkre), akkor küldjünk egy levelet a Documentation Engineering Team <doceng@FreeBSD.org> címére és segítenek megszervezni az állomány átvételét.

Akármelyik megoldást is választjuk, a send-pr(1) használatával ne felejtsük el jelezni, hogy beküldtük a fordítást. Mivel nem nagyon valószínű, hogy a fordítást ténylegesen tároló személy folyékonyan beszélné az adott nyelvet, ezért mielőtt elküldjük, érdemes rendesen átnézetni a fordításunkat.

Valaki (valószínűleg a Dokumentációs Projekt valamelyik vezetője, a Documentation Engineering Team <doceng@FreeBSD.org> tagja) ezután ellenőrzi, hogy beküldött fordítás nem tartalmaz semmilyen technikai hibát. Különösen a következőkre figyelnek ilyenkor:

1. Minden állományban megvan a verziókezeléshez szükséges azonosító (mint például a Projekt esetében a „FreeBSD”)?

2. Az `sv_SE.ISO8859-1` könyvtárban hibamentesen lefut a `make all` parancs?

3. A fordítással kiegészítve a teljesen FreeBSD dokumentációra hibamentesen lefut a `make install` parancs?

Akárki is nézi majd át a beküldött fordítást, ha az előbb felsoroltak bármelyikével probléma akad, vissza fogják küldeni, hogy javítsuk ki.

Ha viszont mindent rendben találnak, akkor a fordításunk hamarosan bekerül a FreeBSD repositoryjába.

K: A fordítás tartalmazhat a nyelvre vagy országra vonatkozó további információkat?

V: Alapvetően ezt nem javasoljuk.

Például a kézikönyv koreai fordításában szeretnénk hozzáadni egy szakaszt a Koreában található boltokról.

Igazából nem látjuk indokoltnak, hogy ez az információ miért ne lehetne része az angol (vagy német, spanyol, magyar stb.) változatoknak. Könnyen előfordulhat ugyanis, hogy egy Koreában elő, angol nyelvi beszélő szeretne a környéken keresni egy ilyen üzletet. Mellesleg ezzel inkább jobban láthatóvá válik mindenki számára, hogy a FreeBSD a világ mennyi országában elérhető. Ez azért nem is olyan rossz.

Ha tehát valamilyen országfüggő információt szeretnénk betenni a dokumentációba, akkor először küldjünk róla egy hibajelentést (a send-pr(1) segítségével) a kézikönyv számára, és csak ezután fordítsuk vissza az adott nyelvre.

Köszönjük az együttműködést!

K: Hogyan illeszthetőek be a dokumentációba nemzeti karakterek?

V: Az alap ASCII készletben meg nem jelenő karaktereket hivatalosan SGML egyedek formájában kell használnunk.

Röviden: egy „és jellel” (&) kezdődnek, majd az azonosítójukat követően egy pontosvesszővel (;) zárulnak.

A nemzeti karakterek ábrázolására használható egyedeket az ISO8879 szabványban definiálták, amely a Portgyűjteményből a textproc/iso8879 porton keresztül érhető el.

Néhány példaképpen ezek közül:
Egyed: é
Megjelenés: é
Leírás: Ékezetes „e”.
Egyed: É
Megjelenés: É
Leírás: Ékezetes „E”.
Egyed: ü
Megjelenés: ü
Leírás: Trémás (umlautos, kétpontos) „u”.

Miután telepítettük az említett portot, a `/usr/local/share/xml/iso8879` könyvtárban található állományokban lesz a szabvány szerint elfogadott összes egyed.

K: Hogyan szólítsuk meg az Olvasót?

V: Az angol nyelvű dokumentációban az Olvasót általában a „you” szóval szokták megszólítani, azonban ezzel sok más nyelvtől eltérően nem választják külön az informális és formális stílust.

Ha olyan nyelvre fordítunk, amelyben létezik ez a megkülönböztetés, próbáljunk az adott nyelven írott szakszövegek stílusához illeszkedni. Ha nincs semmilyen ötletünk, akkor írjunk visszafogott, illedelmes megfogalmazásban.

K: Kell más egyéb információt elhelyeznünk a fordításokban?

V: Igen!

Az angol nyelvű dokumentumok fejléce általában valahogy így szokott kinézni:

```
<!--
     The FreeBSD Documentation Project
```

```
     $FreeBSD: doc/en_US.ISO8859-1/books/fdp-primer/translations/
chapter.xml,v 1.5 2000/07/07 18:38:38 dannyboy Exp $
-->
```

A pontos felépítés ettől némileg eltérhet, de szinte biztos, hogy mindig találunk benne egy $FreeBSD$ kezdetű sort, illetve egy `The FreeBSD Documentation Project` szöveget. A $FreeBSD$ részt a verziókezelő rendszer fogja magától behelyettesíteni, ezért az új állományok esetében ennek üresnek kell lennie (egyszerűen csak `$FreeBSD$`).

A fordításoknak tartalmazniuk kell egy saját $FreeBSD$ sort, illetve a `FreeBSD Documentation Project` nevet cseréljük ki az adott nyelvhez tartozó `The FreeBSD nyelv-angolul Documentation Project` névre.

Mindezek mellett érdemes még egy harmadik sort is felvenni a dokumentumba, amellyel jelezzük a forráskódban, hogy a fordítás melyik angol nyelvű szöveg alapján készült.

Ennek megfelelően tehát a magyar fordításokban általában a következő szöveg szerepel:

```
<!--
     The FreeBSD Hungarian Documentation Project

     $FreeBSD: head/hu_HU.ISO8859-2/books/fdp-primer/
translations/chapter.xml 43126 2013-11-07 16:37:11Z gabor $
     Original revision: 1.31
-->
```

10. fejezet - A fogalmazás stílusa

A FreeBSD dokumentációját készítő rengeteg író munkájának összehangolására ebben a fejezetben megadunk néhány követendő alapelvet.

Az angol nyelvű dokumentáció írásakor az amerikai angol szerinti helyesírást használjuk!
: A szavak helyesírását tekintve az angolnak több különböző változata létezik. Vitás helyzetekben az egységesség kedvéért ezért mindig az amerikai helyesírást tekintsük irányadónak. Ennek megfelelően tehát „color” és nem „colour”, „rationalize” és nem „rationalise”, stb.

Megjegyzés

A brit angol használata elfogadott lehet beküldött cikkek esetében, viszont ilyenkor a helyesírásnak egységesnek kell lennie a teljes dokumentumon belül. Az összes többi dokumentum, tehát könyvek, honlapok, man oldalak stb. esetén azonban mindig amerikai angolt kell alkalmazni.

Ne rövidítsünk!
: Ne alkalmazzunk rövidítéseket a szövegben. Mindig minden kifejezést, szót írjunk ki teljes alakjában. „Pl. ez a példa” tehát nem helyes. Angol nyelven mindez az összevonások elkerülésére vonatkozik, tehát a formális fogalmazási stílusra.

  A rövidítések elhagyásával könnyebb a szövegnek formális jelleget adni, így sokkal precízebben megfogalmazott, a fordítók számára is érthetőbb mondatokat nyerünk.

A felsorolásoknál tegyünk ki vesszőket!
: Angol nyelven, ha több elemet sorolunk fel egyetlen bekezdésben, akkor ezeket mindig vesszőkkel kell tagolnunk. Az utolsó elemnél mindezt egészítsük ki még egy „and” („és”) szóval. A magyarban figyeljünk arra, hogy ez elé már nem kell vessző.

  Például tekintsük a következő mondatot:

  > This is a list of one, two and three items.

  Magyarul:

  > Ez a lista egy, két és három elemből áll.

  Az angol változat esetén felvetődhet a kérdés, hogy ez a lista most „egy”, „két” és „három” elemből áll, vagy „egy”, „két és három” elemből.

Ezért itt az utolsó tag előtt is ki kell tenni a vesszőt:

> This is a list of one, two, and three items.

Kerüljük a szóismétlést!

Lehetőség szerint törekedjünk a szóismétlések elkerülésére. Ez konkrétan a „a parancs”, „az állomány” és „man parancs” jellegű kifejezések mellőzését jelenti, mert ezek sokszor feleslegesen szerepelnek a szövegben. A magyar fordításban azonban néha hasznosnak bizonyulnak, különösen a ragozásban.

Most mutatunk két példát a parancsokra. Ezek közül a másodikban bemutatott stílust javasoljuk az angol szövegek esetén.

Use the command `cvsup` to update your sources.

Use `cvsup` to update your sources.

A magyar szövegben viszont ennek tökéletesen elfogadott a következő típusú fordítása, mivel így könnyebb ragozni a parancsot:

A forrásainkat a `cvsup` paranccsal frissítsük.

Ha a magyarban is el akarjuk kerülni minden áron az ilyen jellegű ismétléseket, akkor próbálkozhatunk úgy írni a mondatot, hogy ne kelljen az idegen szót ragoznunk:

A `cvsup` segítségével frissítsük a forrásainkat.

Az alábbi példákban az állományok neveire láthatunk példákat, amelyek közül ismét a másodikat javasoljuk az angol nyelv esetén:

… in the filename `/etc/rc.local` …

… in `/etc/rc.local` …

A magyarban szintén a korábbiak érvényesek.

A most következő példákban man hivatkozásokat láthatunk. Közülük ismét a második lesz a javasolt:

See `man csh` for more information.

See csh(1).

A magyar fordításban:

Lásd csh(1).

Vagy:

Lásd a csh(1) man oldalt.

Mindig hagyjunk két szóközt a mondatok között!

A mondatok végén mindig hagyjunk két szóköznyi helyet. Ezáltal javul a szöveg olvashatósága, valamint megkönnyíti az Emacs és a hozzá hasonló eszközök használatát.

Habár vitatható, hogy ez a megkülönböztetés egyáltalán szükséges-e, bizonyos esetekben valóban hasznos lehet, különösen neveknél. Erre remek példa „Jordan K. Hubbard”. Ebben a névben középen található egy `H`, amelyet a mondat végéhez hasonlóan egy pont és egy szóköz követ, viszont jól látható, hogy itt nem ér véget a mondat.

Az angol nyelvű fogalmazási stílus szabályairól részletesebb bemutatást William Strunk Elements of Style című könyvéből kaphatunk.

10.1. A forráskód stílusa

Mivel a dokumentáció forrását egyszerre többen szerkesztik, valamilyen módon egységes formában kell tartani. Ennek érdekében legyünk szívesek az alábbiakban megadott iránymutatások szerint dolgozni.

10.1.1. Kis- és nagybetűk

A címkéket *soha ne* nagybetűkkel, hanem mindig kisbetűkkel írjuk, például `para` és *nem* `PARA`.

Az SGML környezetekben megjelenő szövegeket viszont általában nagybetűvel kell írni, például `<!ENTITY…>` , `<!DOCTYPE…>` , és *nem* `<!entity…>` vagy `<!doctype…>` .

10.1.2. Mozaikszavak

A mozaikszavakat első alkalommal általában illik rendesen kiírni, például: „Network Time Protocol (NTP)”. Miután definiáltuk a mozaikszó mögött álló jelentést, elegendő csak a rövidített alakot használni (nem kell tehát a teljes kifejezést, kivéve, ha az adott szövegkörnyezetben annak több értelme van). A mozaikszavakat dokumentumonként egyszer definiáljuk. Ha viszont nekünk jobban megfelel, akkor akár fejezetenként is kifejthetjük az egyes mozaikszavakat.

A mozaikszavak első három megjelenését az `acronym` elemmel kell jelölni, ahol egy `role` tulajdonságban megadjuk a mögött álló teljes kifejezést. Ennek köszönhetően a dokumentumok feldolgozása során létre lehet hozni szószedetet az alkalmazott rövidítésekhez, illetve a honlapokon meg lehet oldani, hogy ha az egérrel felé visszük a kurzort, megjelenjen a teljes megnevezés.

10.1.3. Tördelés

Mindegyik forrás tördelése a nulladik oszloptól indul, *függetlenül* attól, hogy az adott állományt milyen más állomány fogja később tartalmazni.

A nyitócímkék után két szóközzel kell bentebb húzni a szöveget. Ennek megfelelően a zárócímkék pedig két szóközzel csökkentik az aktuális behúzás mértékét. A sorok elején szereplő szóközöket nyolcas csoportban cseréljünk tabulátorokra. Ne használjunk szóközöket a tabulátorok előtt, és ne tegyünk további szóközöket a sorok végére. Ha az elemek tartalma egy sornál hosszabb, akkor a következő sort az elem nyitócímkéjéhez képest mindig két szóközzel bentebb kell kezdeni.

Például ennek a szakasznak így néz ki a szabályos tördelése:

```
+--- Ez a nulladik oszlop
V
<chapter>
  <title>...</title>

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

    <sect2>
      <title>Tördelés</title>

      <para>Mindegyik forrás tördelése a nulladik oszloptól indul,
 <emphasis>függetlenül</emphasis> attól, hogy az adott állomány
 milyen más állomány fogja később tartalmazni.</para>

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

Ha az Emacs vagy XEmacs szerkesztőkkel dolgozunk, akkor az állományok megnyitásakor automatikusan be kellene töltődnie az `sgml-mode` kiegészítésnek, illetve az egyes források végén található változók pontosan a fenti szabályok betartatásáról gondoskodnak.

A Vim szerkesztővel dolgozóknak pedig a következő beállításokat javasoljuk:

```
augroup sgmledit
  autocmd FileType sgml set formatoptions=cq2l -" Speciális ↺
formázási beállítások
  autocmd FileType sgml set textwidth=70       -" Legfeljebb 70 ↺
oszlop széles sorok
  autocmd FileType sgml set shiftwidth=2       -" Az automatikus ↺
behúzás mértéke
  autocmd FileType sgml set softtabstop=2      -" A tabulátor 2 ↺
szóközzel visz bentebb
  autocmd FileType sgml set tabstop=8          -" 8 szóköz cseréje ↺
egy tabulátorra
  autocmd FileType sgml set autoindent         -" Automatikus behúzás
```

```
augroup END
```

10.1.4. A címkék stílusa

10.1.4.1. A címkék elrendezése

Az egy behúzási szinten található címkéket mindig válasszuk el egy üres sorral, a többi esetben viszont ne:

```
<article>
  <articleinfo>
    <title>NIS</title>

    <pubdate>1999 október</pubdate>

    <abstract>
      <para>...
 ...
 ...</para>
    </abstract>
  </articleinfo>

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

    <para>...</para>
  </sect1>

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

    <para>...</para>
  </sect1>
</article>
```

10.1.4.2. A címkék tagolása

Bizonyos címkék, mint például az `itemizedlist`, amelyekben további címkék szerepelnek és nem karakteres adat, mindig egyedül állnak egy sorban.

A `para` és `term` címkék esetén viszont szükség van további címkékre a karakteres adatok befoglalásához, ezért ilyenkor a tartalom közvetlenül a címke után következik, *ugyanabban a sorban*.

Ugyanez érvényes az említett címketípusok zárásakor.

A címketípusok keveredése egy nyilvánvaló problémát eredményez.

Amikor egy karakteres adatot tárolni nem képes elemet nyitó címke közvetlenül követ egy karakteres adatokat bevezető címkét, külön sorba kell kerülniük. A második címkét a szabályok szerint kell behúzni.

Amikor egy karakteres adatokat befoglaló címke záródik közvetlenül a karakteres adatokat tartalmazni nem képes címke után, szerepelhetnek ugyanabban a sorban.

10.1.5. Változtatások a forrás tördelésén

A források változtatása során ügyeljünk arra, hogy *sose tároljunk egyszerre a repositoryba tartalmat és tördelést érintő módosításokat*.

Ennek köszönhetően a dokumentációt fordító csapatok könnyebben észreveszik, hogy mi változott a módosításunk nyomán. Így nem kell azon gondolkozniuk, hogy vajon most tényleg változott a tartalom, vagy csak újratördeltük a sorokat.

Például ha felvettünk két mondatot még egy bekezdéshez, és ezzel az adott bekezdés sorainak hossza túlságosan megnőtt, akkor először tároljuk a hosszú sorokat tartalmazó változatot. Ezután végezzük el a szükséges tördelést és tároljuk azt a változatot is. Ez utóbbi esetben azonban ne felejtsük egyértelműen jelezni a tároláshoz tartozó üzenetben, hogy csak a tördelésen változtattunk („whitespace-only change”). Így a fordítók tudni fogják, hogy ezt figyelmen kívül kell hagyniuk.

10.1.6. Nem törhető szóközök

Lehetőleg kerüljük a sortöréseket olyan helyeken, ahol csúnyán néznének ki, vagy rontanának a szöveg olvashatóságán. A sortörések mindig a kimeneti formátum által alkalmazott sorszélességtől függenek. Különösen a HTML oldalakon található formázott bekezdések jelennek meg ízléstelenül egy szöveges böngészőben, mint például ez is:

```
Az adattároló kapacitása általában 40 MB és 15
GB között változik.  Hardveres tömörítéssel …
```

Az ` ` általános egyed viszont megtiltja az egymáshoz szorosan kötődő elemek közti sortörést. Az ilyen „nem törhető” szóközök használatát elsősorban a következő helyeken javasoljuk:

- mennyiségek és egységek között:

  ```
  57600 bps
  ```

- program neve és verziószáma között:

  ```
  FreeBSD 7.1
  ```

- több szóból álló nevek esetén (óvatosan bánjunk ezzel viszont olyan hosszabb neveknél, mint például a „The FreeBSD Brazilian Portugese Documentation Project”):

  ```
  Sun Microsystems
  ```

10.2. Szólista

Ebben a rövid szólistában összefoglalunk néhány angol szót a FreeBSD Dokumentációs Projektben alkalmazandó írásmódjuk szerint. Ha a keresendő szó nem szerepel ebben a felsorolásban, nézzük meg az O'Reilly-féle gyűjteményt.

- 2.2.X
- 4.X-STABLE
- CD-ROM
- DoS *(Denial of Service)*
- Ports Collection
- IPsec
- Internet
- MHz
- Soft Updates
- Unix
- disk label
- email
- file system
- manual page
- mail server
- name server
- null-modem
- web server

11. fejezet - Az `sgml-mode` használata az Emacs szövegszerkesztőben

Az Emacs és XEmacs újabb változataihoz tartozik egy `psgml` nevű, nagyon hasznos csomag (a Portgyűjteményből a editors/psgml portból telepíthetjük fel). Ez a kiegészítés vagy az `.xml` állományok megnyitásakor töltődik be automatikusan, vagy pedig az `M-x sgml-mode` parancs begépelésével. Általánosságban véve ez az SGML állományok és a bennük található elemek és tulajdonságok szerkesztésére alkalmas mód.

Az alábbiakban bemutatunk néhány olyan alapvető parancsot ebben a módban, amelyekkel könnyebbé válik a különböző SGML dokumentumok, többek közt a kézikönyv szerkesztése.

`C-c C-e`
: Meghívja az `sgml-insert-element` függvényt. Ekkor meg kell adnunk az adott pontra beillesztendő elem nevét. Itt a Tab lenyomásával kérhetjük a név kiegészítését, az adott ponton érvénytelen elemek neveit ilyenkor nem érhetjük el.

  A szövegbe ekkor bekerülnek az elemhez tartozó kezdő- és zárócímkék. Amennyiben az elemhez még tartoznak más egyéb kötelező elemek is, akkor egyúttal ezek is beszúródnak.

`C-c =`
: Meghívja az `sgml-change-element-name` függvényt. A parancs használatához álljunk a módosítandó elembe. A végrehajtáshoz meg kell még adnunk azt is, hogy mire akarjuk átírni az elem nevét. Ezután az érintett elem kezdő- és zárócímkéi lecserélődnek.

`C-c C-r`
: Meghívja az `sgml-tag-region` függvényt. A használatához először jelöljük ki a szöveg egy részét (vigyük a kurzort a kijelölés kezdetéhez, adjuk ki a `C-space` billentyűparancsot, vigyük a kurzort a kijelölés végéhez és ismét adjuk ki a `C-space` parancsot). Ezután meg kell adnunk még a bejelölt rész jelöléséhez használni kívánt elemet. Ennek eredményeképpen végül a kijelölt szakasz elejére és végére bekerül az adott elem kezdő- és zárócímkéje.

`C-c -`
: Meghívja az `sgml-untag-element` függvényt. Álljunk a kurzorral az eltávolítani kívánt elem kezdő- vagy zárócímkéjére és adjuk ki a parancsot. Ekkor az elem kezdő- és zárócímkéi törlésre kerülnek.

`C-c C-q`
: Meghívja az `sgml-fill-element` függvényt. Ennek hatására az elem, amelyben állunk a kurzorral rekurzívan feldolgozásra kerül (például újraformázódik). Ez a változtatás a tördelést is érinteni *fogja*, tehát például még `programlisting` elemek esetében is. Ezért mindig csak körültekintéssel alkalmazzuk!

`C-c C-a`
: Meghívja az `sgml-edit-attributes` függvényt. Ekkor a legközelebbi befoglaló elemhez megnyílik egy másik szerkesztési pufferben az összes hozzá tartozó tulajdonság, értékekkel együtt. Itt a Tab lenyomásával tudunk lépkedni az egyes elemek között, a `C-k` paranccsal lecserélni egy meglevő értéket egy újra, illetve a `C-c C-c` paranccsal bezárni a puffert és visszatérni az eredeti dokumentum szerkesztéséhez.

`C-c C-v`
: Meghívja az `sgml-validate` függvényt. Felajánlja a jelenleg megnyitott dokumentum mentését (amennyiben szükséges) és ellenőrzi az SGML szabvány szerinti érvényességét. A vizsgálat eredménye egy új pufferbe kerül, ahol szépen sorban végig tudjuk nézni az összes hibát és javítani ezeket menet közben.

`C-c /`
: Meghívja az `sgml-insert-end-tag` függvényt. Bezárja a kurzor előtt megkezdett elemet.

Nyilvánvalóan ebben a módban még vannak további hasznos funkciók, de az említetteket használják a leggyakrabban.

A Dokumentációs Projekten belüli munkához az `.emacs` állományban a következő bejegyzéseket érdemes megadni a megfelelő tördeléshez, elrendezéshez és sorszélességhez:

```
(defun local-sgml-mode-hook
  (setq fill-column 70
        indent-tabs-mode nil
        next-line-add-newlines nil
        standard-indent 4
        sgml-indent-data t)
  (auto-fill-mode t)
  (setq sgml-catalog-files -'("/usr/local/share/xml/catalog")))
(add-hook -'psgml-mode-hook
  -'(lambda () (local-psgml-mode-hook)))
```

12. fejezet - Lásd még...

Ez a dokumentum szándékosan nem törekszik az SGML és az említett DTD-k, valamint a FreeBSD Dokumentációs Projekt részletes bemutatására. Ezekről részletesebb információkat az ebben a fejezetben összegyűjtött hivatkozások mentén kaphatunk.

12.1. FreeBSD Dokumentációs Projekt

- A FreeBSD Dokumentációs Projekt honlapja
- A FreeBSD kézikönyv

12.2. SGML

- Az SGML/XML honlapja, minden, ami SGML
- Könnyed bevezetés az SGML használatába (angolul)

12.3. HTML

- A World Wide Web Consortium honlapja
- A HTML 4.0 specifikációja

12.4. DocBook

- DocBook Műszaki Bizottság, a DocBook DTD karbantartói
- DocBook: The Definitive Guide, a DocBook DTD interneten olvasható dokumentációja
- Nyílt DocBook Repository, különböző DSSSL stíluslapok és egyéb források a DocBook felhasználók számára

12.5. Linux Dokumentációs Projekt

- A Linux Dokumentációs Projekt honlapja

A. függelék - Példatár

Ebben a függelékben bemutatunk néhány minta SGML forrást, illetve azokat a parancsokat, amelyekkel egyik formátumból a másikba lehet ezeket alakítani. Amennyiben sikeresen telepítettük rendszerünkre a Dokumentációs Projektben használt segédprogramokat, akkor az itt megadott minta forrásokat akár közvetlenül is fel tudjuk használni.

A mintaképp mellékelt források nem fednek le mindent — nem tartalmazzák az összes korábban ismertetett elemet, és egyáltalán nem térnek ki a rövidebb részek, például bevezetés, előszó, köszönetnyilvánítás stb. jelölésére. Ha konkrét jelölési megoldásokra lenne szükségünk, akkor kérjük le a repositoryból a `doc` CVSup gyűjteményt, és nézzük át a benne szereplő SGML forrásokat, vagy böngésszük ezeket közvetlenül a `http://www.FreeBSD.org/cgi/cvsweb.cgi/doc/` honlapon keresztül.

A félreértések elkerülése végett ezek a példák a szabvány DocBook 4.1 DTD szerint íródtak, mellőzik a FreeBSD kiterjesztéseit. Ugyanúgy nem építkeznek a FreeBSD Dokumentációs Projekt által módosított stíluslapokra sem, hanem a Norm Walsh eredetileg kiadott stíluslapjait használják. Ennek köszönhetően általános DocBook mintáknak is tekinthetőek.

A.1. DocBook könyv, a `book` elem

A.1. példa - DocBook `book`

```
<!DOCTYPE book PUBLIC -"-//OASIS//DTD DocBook V4.1//EN">

<book lang='hu'>
  <bookinfo>
    <title>Könyvminta</title>

    <author>
      <surname>Vezetéknév</surname>
      <firstname>Keresztnév</firstname>
      <affiliation>
        <address><email>ize@minta.hu</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2008</year>
      <holder>A copyright szövege</holder>
    </copyright>

    <abstract>
```

```
      <para>Ha tartozik a könyvhöz rövid tartalmi összefoglaló
 (absztrakt), akkor azt ide írjuk.</para>
    </abstract>
  </bookinfo>

  <preface>
    <title>Előszó</title>

    <para>A könyvhöz tartozhat előszó is, amelyet itt kell
      szerepeltetnünk.</para>
  </preface>

  <chapter>
    <title>Első fejezet</title>

    <para>Ez a könyv első fejezetének tartalma.</para>

    <sect1>
      <title>Az első szakasz</title>

      <para>Ez a könyv első szakasza.</para>
    </sect1>
  </chapter>
</book>
```

A.2. DocBook cikk, az `article` elem

A.2. példa - DocBook `article`

```
<!DOCTYPE article PUBLIC -"-//OASIS//DTD DocBook V4.1//EN">

<article lang='hu'>
  <articleinfo>
    <title>Cikkminta</title>

    <author>
      <surname>Vezetéknév</surname>
      <firstname>Keresztnév</firstname>
      <affiliation>
        <address><email>ize@minta.hu</email></address>
      </affiliation>
    </author>

    <copyright>
      <year>2008</year>
      <holder>A copyright szövege</holder>
    </copyright>
```

```
    <abstract>
      <para>Ha tartozik a cikkhez rövid tartalmi összefoglalás
 (absztrakt), akkor annak ide kell kerülnie.</para>
    </abstract>
  </articleinfo>

  <sect1>
    <title>Első szakasz</title>

    <para>Ez a cikk első szakasza.</para>

    <sect2>
      <title>Első alszakasz</title>

      <para>Ez a cikk első alszakasza.</para>
    </sect2>
  </sect1>
</article>
```

A.3. A formázott kimenet előállítása

Ebben a szakaszban feltételezzük, hogy már vagy kézzel vagy pedig a hozzá tartozó port segítségével telepítettük a textproc/docproj portban szereplő segédeszközöket. Emellett továbbá még feltesszük, hogy az összes eszközt a `/usr/local` könyvtár alá telepítettük és a binárisok elérési útvonala része a `PATH` környezeti változónak. Amennyiben ezektől a feltételezésektől valamilyen módon eltértünk, akkor a példákat értelemszerűen a saját környezetünkre alkalmazva kell végrehajtani.

A.3.1. A Jade használata

A.3. példa - DocBook forrás átalakítása HTML formátumúra (egyetlen nagy állomány)

```
% jade --V nochunks \   ❶
    --c -/usr/local/share/xml/docbook/dsssl/modular/catalog \    ❷
    --c -/usr/local/share/xml/docbook/catalog \
    --c -/usr/local/share/xml/jade/catalog \
    --d -/usr/local/share/xml/docbook/dsssl/modular/html/
docbook.dsl \❸
    --t sgml ❹ állomány.xml > állomány.html   ❺
```

❶ A `nochunks` paramétert adja át a stíluslapoknak és az eredményt a szabványos kimenetre irányítattja át (Norm Walsh stíluslapjait használjuk).

❷ Megadjuk a Jade által feldolgozandó katalógusokat. Itt három katalógust kell megadni. Az első katalógus a DSSSL stíluslapok, a második a DocBook DTD és a harmadik a Jade számára tartalmaz információkat.
❸ A Jade a dokumentum feldolgozásához az itt megadott DSSSL stíluslapot fogja felhasználni.
❹ A Jade itt kap utasítást arra, hogy az egyik DTD-ból a másikba *alakítsa át* a dokumentumot. Ebben a példában most a DocBook DTD-ból alakítunk át a HTML DTD-ba.
❺ Megadjuk a feldolgozandó állományt a Jade számára és átirányítjuk a kimenetet egy `.html` kiterjesztésű állományba.

A.4. példa - DocBook forrás átalakítása HTML formátumúra (több kisebb állomány)

```
% jade \
    --c -/usr/local/share/xml/docbook/dsssl/modular/catalog \    ❶
    --c -/usr/local/share/xml/docbook/catalog \
    --c -/usr/local/share/xml/jade/catalog \
    --d -/usr/local/share/xml/docbook/dsssl/modular/html/
docbook.dsl \❷
    --t sgml ❸ állomány.xml  ❹
```

❶ Megadjuk a Jade által feldolgozandó katalógusokat. Itt három katalógust kell megadni. Az első katalógus a DSSSL stíluslapok, a második a DocBook DTD és a harmadik a Jade számára tartalmaz információkat.
❷ A Jade a dokumentum feldolgozásához az itt megadott DSSSL stíluslapot fogja felhasználni.
❸ A Jade itt kap utasítást arra, hogy az egyik DTD-ból a másikba *alakítsa át* a dokumentumot. Ebben a példában most a DocBook DTD-ból alakítunk át a HTML DTD-ba.
❹ Megadjuk a feldolgozandó állományt a Jade számára. A stíluslap fogja majd eldönteni, hogy mi legyen a neve a menet közben keletkező egyes HTML állományoknak, illetve a „gyökérnek” (ez az az állomány, ahonnan a dokumentum kezdődik).

Előfordulhat, hogy ez a parancs szintén csak egyetlen HTML állományt generál. Ez függ a feldolgozandó dokumentum szerkezetétől és a stíluslap feldarabolást vezérlő szabályaitól.

A.5. példa - DocBook forrás átalakítása Postscript formátumúra

Az SGML forrást TeX állománnyá akarjuk alakítani.

```
% jade --V tex-backend \  ❶
    --c -/usr/local/share/xml/docbook/dsssl/modular/catalog \    ❷
    --c -/usr/local/share/xml/docbook/catalog \
    --c -/usr/local/share/xml/jade/catalog \
    --d -/usr/local/share/xml/docbook/dsssl/modular/print/
docbook.dsl \❸
    --t tex ❹ állomány.xml
```

❶ Felparaméterezzük a stíluslapot a TeX formátumú kimenet előállításához.
❷ Megadjuk a Jade által feldolgozandó katalógusokat. Itt három katalógust kell megadni. Az első katalógus a DSSSL stíluslapok, a második a DocBook DTD és a harmadik a Jade számára tartalmaz információkat.
❸ A Jade a dokumentum feldolgozásához az itt megadott DSSSL stíluslapot fogja felhasználni.
❹ Megadjuk a Jade számára, hogy TeX formátumú kimenetet készítsen.

Az így keletkező `.tex` kiterjesztésű állomány aztán a `&jadetex` makrócsomaggal együtt átadható bemenetként a `tex` parancsnak.

```
% tex -"&jadetex" állomány.tex
```

A `tex` parancsot *legalább* háromszor le kell futtatni. Először feldolgozza a dokumentumot, és szétválogatja az egyes részeit, hogy meg tudja állapítani részeit hivatkoztuk valahonnan máshonnan, hogyan indexelje stb.

Ha ebben a fázbisban különböző figyelmeztetéseket látunk, mint például LaTeX Warning: Reference `136' on page 5 undefined on input line 728., akkor még ilyenkor ne foglalkozzunk különösebben velük.

A második futtatás során újra feldolgozza a dokumentumot a korábbi feldolgozásból származó bizonyos előismeretek (például a dokumentum oldalszámának) alapján. Ekkor az indexek és a kereszthivatkozások már gond nélkül feloldhatóak.

A harmadik menetben elvégzi az utolsó simításokat, amennyiben szükség van rájuk.

Ebben a fázisban egy `állomány.dvi` alakú eredményt kapunk.

Végezetül az imént kapott `.dvi` állomány Postscript formátumúra alakításához futtassuk le a `dvips` parancsot:

```
% dvips --o állomány.ps állomány.dvi
```

A.6. példa - DocBook forrás átalakítása PDF formátumúra

A feldolgozási folyamat első része hasonló ahhoz, amikor DocBook forrásból akarunk Postscript formátumú állományt készíteni, tehát elegendő a `jade` parancsot az előbb megadott paraméterekkel meghívni (lásd A.5. példa - DocBook forrás átalakítása Postscript formátumúra).

Amikor viszont megkaptuk a `.tex` állományt, akkor a pdfTeX programot futtassuk le rá. Ügyeljünk arra, hogy ekkor már a `&pdfjadetex` makrócsomagot kell használnunk:

```
% pdftex -"&pdfjadetex" állomány.tex
```

Ebben az esetben is háromszor kell lefuttatnunk a parancsot.

Ennek eredményeképpen aztán végül előáll egy további feldolgozást már nem igénylő `állomány.pdf` állomány.

Tárgymutató