Készítsünk Impress és PowerPoint bemutatót LaTeX és Perl segítségével

Aki valamiért kénytelen védett formátumokat használni, a nyílt forráskód annak is megkönnyítheti a dolgát.

Kezdjük az elején.
Második könyvem, melyet ***Dr. Michael Moorhouse***-al közösen készítettünk, végre elkészült. Hat hónap plusz időt töltöttem vele, ami egyben azt is jelentette, hogy hat hónap késésben voltam. Az összes időt szedéssel, átolvasással, írással és ***Michael Microsoft Word*** állományainak ***LaTeX*** formátumúra alakításával töltöttem, elolvasva majd újra elolvasva mindent. Végül mindet még egyszer átnéztem. Amikor végre minden elkészült, teljesen kimerültem. Nem sokkal ez után megkaptam a borító végleges mintáját. És akkor kiderült, hogy a hátlapon azt ígérjük, hogy a weblapon a szöveggel együtt használható ***Microsoft PowerPoint*** bemutatók jelennek majd meg. A borító megváltoztatásához már túlságosan késő volt, következésképpen valahogy meg kellett csinálni a bemutatókat. Úgy tűnik elfelejtettem, hogy ebben egyeztünk meg a projekt indításakor, több mint 18 hónappal korábban.

A PowerPoint „szabvány"

Nyolc hónappal ezelőtt, akadémia berkekben a bemutatók de facto szabvány formátuma a ***PowerPoint*** volt. Manapság már a ***PDF*** is népszerű. Mint oly sokan a Linux közösségben, már régebben átálltam az ***OpenOffice.org***-ra és elhagytam a ***PowerPoint*** környeztet. A könyvben 20 fejezet szerepelt, úgy becsültem, hogy körülbelül 20 napba telik elkészíteni a diákat. Ráadásul mindezt a ***PowerPointtal*** megcsinálni nem igazán volt ínyemre. Természetesen dolgozhattam volna az ***OpenOffice.org Impress***-ében, majd azt exportálhattam volna ***PowerPointba***, de ez az ötlet sem tetszett különösebben. Az alapvető probléma az volt, hogy jól tudtam, minden adat megvolt ***LaTeX*** állományokban, és ha arra gondoltam, hogy ezeket mind újra létre kell hozni egy fóliakészítő alkalmazásban csak még kimerültebbnek éreztem magam. Ha valahogy ki tudnám találni, hogyan lehet programozottan kiszedni az adatokat a ***LaTeX*** állományaimból és beletenni a ***PowerPoint*** diákba, az jelentősen leegyszerűsítené a dolgokat.

Munka a bemutató fájlformátumokkal

A ***Google*** keresés nem hozott eredményt. Talán nem is meglepő, hogy a ***PowerPoint*** fájlformátum részleteihez igen nehéz ráakadni. Találtam egy ***Microsoft Windows Help*** formátumú állományt amely a ***Microsoft Office*** dokumentumok ***XML*** szabványát ismertette. Ebbe a formátumba lehet a ***PowerPoint*** dokumentumokat exportálni. Sajnos igen méretes és bonyolult írás volt. Miután láttam, hogy a ***Google***-lel nem sokra megyek, átnyergeltem a ***Comprehensive Perl Archive Network (CPAN)*** hálózatára. A ***Perl***, a választott programozási nyelvem, szinte minden fájlformátumra és számítási platformra felkészült. Ha valaki korábban már játszadozott kicsit a ***Perl*** és ***PowerPoint*** párossal, munkájára esetleg ráakadhatok a ***CPAN***-on. Sajnos ez a keresés is eredménytelen maradt.
És akkor rájöttem a megoldásra: ha a nyílt és széles körűen dokumentált ***OpenOffice.org Impress*** dokumentumformátummal tudnék dolgozni, utolsó lépésként az ***Impress*** diákat kimenthetném ***PowerPoint*** formátumba.
Az ***OpenOffice.org*** weblap gyors átolvasása után hamarosan meg is találtam a ***OpenOffice.org*** fájlformátumainak ***XML*** leírását. A szabvány a maga 600 oldalával, súlyosabb mint a saját könyvem!
Az ***XML*** dokumentum szépen meg van írva, de egy kicsit nehéz olvasmány. Visszakanyarodtam hát a ***CPAN***-ra, hátha dolgozott már valamelyik programozó az ***OpenOffice.org*** formátumaival, és volt olyan nagylelkű, hogy feltöltötte munkáját a ***CPAN***-ra. Ezúttal nem kellett csalódnom. ***Jean-Marie Gouarne*** a ***Genicorptól*** nemrégiben adta közre az ***OpenOffice::OODoc*** modult, amely a ***Perl*** csatoló felületet biztosít az ***OpenOffice.org*** formátumához. Egy már létező dokumentum esetében az ***OpenOffice::OODoc*** modullal módosíthatjuk a tartalmat, igény szerint hozzáadhatunk, törölhetünk és frissíthetünk a lemezes állományban.

A diakészítő terve

Egy egyszerű ***Perl*** szűrővel kezdtem amely ***LaTeX*** állományokat fogad bemenetként és dia adatokat készít kimenetként saját szöveges formátumban. A szöveges állomány létrehozásával biztosítani tudtam, hogy bármilyen szövegszerkesztővel könnyedén módosítani tudom a szűrő kimenetét, illetve szükség esetén finomhangolhatom a szöveges kimenetet. Amikor elértem a kívánt eredményt, egy

másik, szintén *Perlben* készült szűrő a szöveges állományból elkészíti az *Impress* bemutatót. Az *Impress* bemutatót aztán meg lehet nyitni *Impress*-ben és exportálni *PowerPoint* vagy *PDF* formátumban.

Dia tervek

A bemutatómat tudatosan próbáltam a lehető legegyszerűbb alakban elkészíteni, ezért úgy döntöttem csak három diatípust használok. A cím dia tartalmazza a fejezet címét a bemutató állomány elején. A bemutatóban a `title_slide` (cím dia) kettős szerepet tölt majd be, ez tartalmazza majd a bemutatóba illesztett képeket is, azaz képenként külön `title_slide`-ot készítünk. A `bullet_slide` (pont dia) tartalmazza a szakaszok címeit a dia fejlécében valamint az alcímeket a pontokban. Végül, a `sourcecode_slide` típusú, rögzített karakterhosszúságú, betű szerinti diákat használom fel a programforrások megjelenítéséhez.
A három diás bemutatót kézzel készítettem el *Impress*-ben és *blank.sxi*-nek néven mentettem el. Az elkészített három dia az előző bekezdésben bemutatott három diatípusnak felelt meg. Úgy terveztem, hogy a fejezetek programozott bemutatókészítése során ezeket fogom majd másolni. A másolás segítségével biztosíthatom a bemutatóm egységes külalakját.

A szöveges tartalom kigyűjtését végző szűrő

A `getcontent` olyan stílusú héjprogram, amilyeneket igen gyakran készítenek a *Perl* programozók, felhasználják, majd eldobják őket. (A hálózati források közt megtaláljuk a cikkben említett programok forrását) Soronként olvasva végiglépked a szabványos bemeneten, és mintakereséssel próbálja meg felderíteni a lényeges sorokat. Ha a sor megfelel a keresésnek, elkészíti a megfelelő kimenetet.
A `getcontent` működésének bemutatására álljon itt példaként a *LaTeX* állomány fejezetcímeit kezelő kód:

```
if ( /\\chapter\{(.*)\}/ )
{
    print "CHAPTERTITLE: $1\n";
    next;
}
```

A *LaTeX* fejezet makróját egyszerű szabályos kifejezés keresi ki; Ha volt egyezés, a fejezet címét kigyűjtjük és elkészítjük a kimenetet. A `next` hívás rövidre zárja a kört, így ha találtunk valamit, a szabványos bemenetről a következő sort olvassuk be. Ezáltal a következő *LaTeX* részlet:

```
\chapter{Working with Regular Expressions}
```

a következő szöveges tartalomra cserélődik le:

```
CHAPTERTITLE: Working with Regular Expressions
```

Azaz a *LaTeX* jelrendszerét eltávolítottuk, és egy sokkal egyszerűbb jelöléssel cseréljük le. A alfejezet és szakaszcímeket jelölő *LaTeX* formátumot ugyanilyen módszerrel kezeljük. A kód a következőképpen néz ki:

```
if ( /\\section\{(.*)\}/ )
{
    print "BULLETTITLE: $1\n";
    next;
}
if ( /\\subsection\{(.*)\}/ )
{
    print "BULLETCONTENT: $1\n";
    next;
}
```

A forráskódlistákkal sem sokkal nehezebb boldogulni, igaz itt figyelnünk kell rá, hol kezdődik és végződik a nyers szöveg bevitele. A következő kódrészlet kezeli a *LaTeX „verbatim"* blokkját:

```
if ( /\\begin\{verbatim\}/ )
{
    print "STARTCODE\n";
    $in_verbatim = TRUE;
    next;
}
```

A `verbatim` blokkból kilépést pedig a következő kóddal keressük:

```
if ( $in_verbatim )
{
    if ( /\\end\{verbatim\}/ )
    {
        print "STOPCODE\n";
        $in_verbatim = FALSE;
    }
    else
    {
        print;
    }
    next;
}
```

A `$in_verbatim` nevű egyszerű `bool` típusú változó segítségével vizsgáljuk, hogy a parancsfájlunk még a `verbatim` blokk belsejében van-e. Hasonló kód kezeli a könyvben megjelenő tételeket a képeket, azok feliratai és egyéb érdekes dolgokat pedig néhány feltétles blokk kezeli. vegyük például a következő *LaTeX* leírást:

```
\chapter{The Basics}
\textit{Getting started with Perl.}
\section{Let's Get Started!}
There is no substitute for practical experience
➥ when first
learning how to program. So, here is the first
➥ Perl program
\index{welcome@\texttt{welcome}, and the first
➥ program, called
\texttt{welcome}:
\begin{verbatim}
    print "Welcome to the World of Perl!\n";
\end{verbatim}
\noindent When executed by \texttt{perl}
\footnote{We will learn how to do this is in
```

```
just a moment.}, this small program displays
the following, perhaps rather not unexpected,
message on screen:
\begin{verbatim}
    Welcome to the World of Perl!
\end{verbatim}
```

A `getcontent` parancsfájlja a fenti ***LaTeX*** állományból a következő szöveges állományt készíti:

```
CHAPTERTITLE: The Basics
CHAPTERCONTENT: Getting started with Perl.
BULLETTITLE: Let's Get Started!
STARTCODE
    print "Welcome to the World of Perl!\n";
STOPCODE
STARTCODE
    Welcome to the World of Perl!
STOPCODE
```

Figyeljük meg hogy valamennyi ***LaTeX*** jelölés eltűnt, és helyére egy egyszerű jelölésrendszer került, amelyet majd a diák létrehozására használhatunk fel. Feltételezve, hogy a ***LaTeX*** részlet a `chapter3.tex` nevű állományban kapott helyet, a `getcontent` parancsfájlt elindítva, az átalakítás eredményét átirányítjuk a megfelelően elnevezett célállományba:

```
perl getcontent chapter3.tex > chapter3.input
```

A `chapter3.input` állományba így a szöveges tartalom kerül, amit aztán tetszőleges szövegszerkesztővel finomhangolhatunk a diák létrehozása előtt.

Az Impress bemutató készítő szűrő

A diák létrehozását az ***Impress*** dokumentumban több tényező is nehezítette. Először is az ***OpenOffice::OODoc*** modullal nem hozhatunk létre új ***OpenOffice.org*** állományt, az ugyanis csak a már létező állományokat tudja módosítani. Továbbá a modult elsősorban az ***OpenOffice.org Writer*** állományok (tehát szövegszerkesztő fájlok és nem ***Impress*** bemutatók) szerkesztésére készítették. Példaként álljon itt egy rövid, `appendpara` nevű program, amely rövid szöveget fűz a ***Writer*** dokumentumunkhoz:

```
#! /usr/bin/perl -w
use strict;
use OpenOffice::OODoc;
my $document = ooDocument( file => 'blank.sxw' );
$document->appendParagraph
(
    text    => 'Some new text',
    style   => 'Text body'
);
$document->save;
```

A rövid program az `OpenOffice::OODoc` modul segítségével készít dokumentum objektumot egy már meglévő ***Writer*** állományból. A program ezután az `appendParagraph` metódus meghívásával beszúr némi szöveget majd meghívja a `save` metódust és a lemezre menti a változtatásokat.
Az `appendParagraph` metóduson felül az `OpenOffice::OODoc` modulban találunk egy `insertElement` nevű metódust is, amellyel egy adott típusú lapot szúrhatunk a dokumentumba. A lap egy már meglévő lap másolata vagy tényleges nyers ***XML*** lehet.
Alig jutottam el a 6. oldalig a több mint 600 oldalas ***OpenOffice.org XML*** fájlformátum leírásban, máris megtaláltam, hogy az ***Impress*** a `//draw:page` ***XML*** típust használja a bemutatók diáinak megjelenítésére. Sajnos az `OpenOffice::OODoc` modul ezzel az objektumtípussal közvetlenül nem tudott dolgozni, úgyhogy valamilyen más módszert kellett kieszelnem az adatok kezeléséhez.
Egészen pontosan a ***blank.sxi*** dokumentumban lévő lapokat szerettem volna kiemelni és szükség szerint másolni az oldalait, miközben a dia tartalmát a ***getcontent*** parancsfájl által gyűjtött tartalommal helyettesítem. Ehhez persze egy kicsit jobban meg kellett ismernem az ***Impress XML*** formátumát.
Két választással volt: tovább olvasom a 600+ oldalas szabványdokumentációt, vagy belenézek egy létező állományba hátha eleget megtudok a feladat kivitelezéséhez. Az utóbbit választottam. Emlékezve egy korábbi ***Linux Journal*** cikkre mely szerint az ***OpenOffice.org*** több részből álló állományait a népszerű ***ZIP*** algoritmussal tömöríti, készítettem egy ideiglenes könyvtárat és kitömörítettem a ***blank.sxi*** állományt:

```
mkdir unzipped
cd unzipped
unzip ../blank.sxi
```

Ezáltal kaptam néhány fájlt és könyvtárat:

```
content.xml
META-INF
meta.xml
mimetype
settings.xml
styles.xml
```

a legérdekesebb a ***content.xml*** állomány, hiszen ez tartalmazza a dokumentumot alkotó tényleges szöveget. képernyőn vagy egy szövegszerkesztőben nézve rengeteg nehezen értelmezhető ***XML*** kódot látunk. Azért, hogy a részek lehető legkisebbek legyenek, az ***XML*** formázására egyáltalán nem fordítottak semmilyen figyelmet, a zippelt állomány egyik részében sem. Az ***XML*** általában tagolás és szóközök nélküli szövegfolyamként kerül mentésre. Mielőtt értelmezhettem volna, valamilyen olvasható formában kellett kinyomtatnom ezt az ***XML***-t. Pillanatnyi ihlettől vezérelve, átléptem parancssorba és begépeltem az `xml` szót két tabbal kísérve. A képernyőn megjelentek az `xml`-el kezdődő előre telepített szóközök:

```
xml2-config     xml-config      xmllint
xmlto           xml2man         xml-i18n-toolize
xmlproc_parse   xmlwf           xml2pot
xmlif           xmlproc_val     xmlcatalog
xmlizer         xmltex
```

Az `xmllint` egyből felkeltette az érdeklődésemet. A kézikönyv oldalán hamar rábukkantam a `--format` kapcsolóra, amely – ahogy azt biztosan kitalálták – formázza az eszköznek átadott *XML* adatokat. Következésképpen az `xmllint --format content.xml` paranccsal olyan kimenetet kaptam amit aztán a less-be csövezve már nehézségek nélkül el lehetett olvasni. Alább bemutatjuk a *content.xml* olvasható formában kinyomtatott rövidített részletét, ahol a *blank.sxi Impress* dokumentum `title_slide` *XML* részletét láthatjuk:

```
<draw:page draw:name="page1" draw:style- ...
  <draw:text-box presentation:style-name= ...
    <text:p text:style-name="P1">
      <text:span text:style-name="T1">
        ChapterTitleSlide
      </text:span>
    </text:p>
  </draw:text-box>
  <draw:text-box presentation:style-name= ...
    <text:p text:style-name="P3">
      <text:span text:style-name="T2">
        ChapterTitleSlideText
      </text:span>
    </text:p>
  </draw:text-box>
  <presentation:notes>
    <draw:page-thumbnail draw:style-name= ...
    <draw:text-box presentation:style-name ...
  </presentation:notes>
</draw:page>
```

Figyeljük meg a `ChapterTitleSlide` és a `ChapterTitleSlideText` tartalmat, amelyeket a *blank.sxi* létrehozásakor gépeltem be az *Impress*-be. Ha az `insertElement` metódus segítségével e részlet alapján be tudnék vinni nyers *XML* adatokat, miközben az üres tartalmat a saját szöveges adataimmal helyettesítem, lényegében már készen is volnék.
A példa kedvéért gondoljuk végig mi is történik, amikor a bemutató címét és alcímét feldolgozza a `produce_slides` rutin. Az `insertElement` a következő formában hívódik meg, létrehozva a az új diát:

```
$presentation->insertElement( '//draw:page',
  $last_slide++,
    title_slide( $title_title, $title_content ),
      position => 'after' );
```

A `title_slide` alprogram nyers *XML* adatot ad vissza, amit a dokumentumba szúrhatunk.
Ha a `getcontent` által készített szöveges tartalommal egyező formátumú bemenetei állományt használunk, a `produce_slides` parancsfájl lemásolja a *blank.sxi Impress* állományt majd tetszőleges számú diával tölti fel, így programozottan hozza létre a bemutatót. A parancsfájl szerkezetében nem különbözik sokban a `getcontent`-től mindössze annyi a különlegessége, hogy a *blank.sxi* állományban tárolt, három különféle diatípushoz tartozó nyers *XML* adatot átemeli. Az előadás elkészítéséhez a következő alakban hívjuk meg a `produce_slides` programot:

```
perl produce_slides 3 chapter3.input
```

Eredményképpen egy új *Impress* dokumentumot kapunk, amely *chapter3.sxi* néven jelenik meg a merevlemezünkön.
Miután létrehoztam az *Impress* állományt, a grafikus képek helyfoglaló elemeit le kellett cserélnem a tényleges képekre. A `getcontent` parancsfájl azonban csak a képek neveit bontotta ki, és nem magukat a képeket. A képek importálása *Impress* alá nem lenne túl nehéz feladat, eltekintve attól, hogy az általam birtokolt eredeti állományok meglehetősen gyenge minőségűek voltak azokhoz képest amelyek végül a könyvbe kerültek.
A könyvbe kerülő képeket jelentősen feljavította a kiadó végső szedőfázisa. Nekem pedig természetesen nem voltak meg ezek az állományok.
Aztán eszembe jutott, hogy a kiadó elküldte nekem a végső *PDF* változatot, amelyben ott volt valamennyi jó minőségű kép. Az *xpdf* segítségével 200%-ra nagyítottam a képeket, majd beindítva a *The GIMP*-et leolvastam az *xpdf* megjelenítő ablakát. Kivágtam a grafikus képeket és elmentettem *JPEG* formátumban. Igaz beletelt egy kis időbe, amikor azonban elkészült, gyönyörű, könyv-minőségű képekhez jutottam, amelyeket már be lehetett importálni az *Impress* bemutatóba. Miután ezzel elkészültem, elmentettem az *Impress* dokumentumot *PowerPoint* formátumban és a feladat meg volt oldva. A kezdeti 20 napos becslésem körülbelül 20 órás tényleges munkára csökkent.
Ráadásul most már, ha gyorsan kell diákat összeraknom, a szöveges tartalmat akár kézzel, *vi* segítségével is összerakhatom, majd a `produce_slides` parancsfájl lefuttatása után már készen is vagyok.

Zárszó

Ami először lehetetlen feladatnak tűnt – programozottan létrehozni egy *PowerPoint* bemutatót – kiderült, hogy a nyílt forráskódnak köszönhetően nagyon is lehetséges. A megszokott *Red Hat 9* terjesztésemben valamennyi szükséges eszköz rendelkezésemre állt: `vi`, `unzip`, *Perl*, `xmllint`, `xpdf`, *The GIMP* és az *OpenOffice.org* csomag.

Linux Journal 2005. április, 132. szám

Paul Barry (paul.barry@itcarlow.ie)
A Carlow-i Műszaki Intézetben tart előadásokat, Írországban. Weblapján előadásaival, könyveivel és cikkeivel kapcsolatos információkat találunk: glasnost.itcarlow.ie/~barryp.

KAPCSOLÓDÓ CÍMEK

- ftp.ssc.com/pub/lj/listings/issue132/7972.tgz
- search.cpan.org
- www.openoffice.org