LIBREOFFICE WRITER FÁJLFORMÁTUM PROBLÉMÁK ÉS MEGOLDÁSUK

Vajna Miklós

Kivonat

A LibreOffice Writer import és export szűrői egy olyan aránylag jól elkülöníthető terület, ahol kis javításokkal is látványos eredményeket lehet elérni. Ennek mikéntjéről ad rövid áttekintést az alábbi írás.

Tartalomjegyzék

1. Bevezetés

A tavalyi „Hogyan készítsünk új Writer funkciókat” előadás végigjárta egy új funkció legfőbb lépéseit: ezek közül csak egyetlen volt az import és export szűrők kiterjesztése. Sok embernek viszont az az általános véleménye a Writer komponensről, hogy az önmagában, csak ODF formátumot használva úgymond már készen van: amin sokat kéne csiszolni az az egyéb idegen formátumok támogatása. Annyiban biztosan megállapodhatunk, hogy van még hova fejlődni. Ennek érdekében az előadás és a workshop során áttekintést adunk, hogy hogyan lehet olyan problémákat javítani, ami csak az import és/vagy export szűrőket érinti, tehát a dokumentum modellje, a megjelenítés, a felhasználói felület, a súgó, stb. mind rendben van, csak egy adott formátumba mentés, vagy az abból való betöltés nem működik megfelelően.

További változás e területen, hogy tradicionálisan ilyen javításokat önmagukban készítettek el korábban, mindenféle automatikus teszt nélkül. Ennek eredményeként az új fejlesztők féltek a régi kódhoz hozzányúlni, hiszen nem tudták, milyen eseteket tesznek tönkre egy új javítással: nem örültek a karbantartók a nagyobb átalakításoknak. Ezt orvosolandó, az elmúlt években fokozatosan bevezetésre kerültek különféle tesztek a LibreOffice-ban. A mi szempontunkból most a Writer szűrők tesztjei a fontosak: ezeket manapság már annyira könnyű megírni a legtöbb esetben, hogy ritka az olyan javítás, amit a hozzá tartozó tesztek nélkül fogadnak el a fejlesztők.

Ez jó hír abból a szempontból, hogy a javításunk után magunknak és automatikusan le tudjuk futtatni a teszteket, és ha ezek nem jeleznek hibát, jó eséllyel nem okoztunk véletlen mellékhatásokat. Rossz hír, hogy nekünk is teszteket kell írni, ha javítást készítünk – ezért az utolsó fejezet erre is kitér.

A fentiek alapján tehát kijelenthetjük, hogy a fájlformátum problémák egy jól tesztelhető terület, ahol ma már van esély, hogyha egy problémát kijavítunk, az többször nem fordul elő. Hozzá kell azonban tenni, hogy a legtöbb ilyen probléma megoldása a forráskód módosításával történik: ritka az olyan eset, amikor a bemenő dokumentum „kézi” módosításával bármit trükközhetünk.

2. Mi az a fájlformátum probléma?

A fájlformátum probléma tehát egy olyan hiba, amit egy adott formátum import vagy export szűrője okoz.

Jó példa erre:

- Egy vonal vastagsága túl nagy, ha egy csoportos alakzat részét képezi, DOCX megnyitása esetén.
- Ez a dokumentum egy oldalas kéne legyen, nem kettő.

Rossz példák:

- A betöltött dokumentum megjelenítése fagyást (hang vagy crash) okoz.
- A Writer nem támogat egy adott funkciót, amit a fájlformátum viszont támogatna.

Érdemes még megérteni madártávlatból az elején a szűrők működését. Valójában mindig csak exportálunk vagy importálunk. A többi eset mind ezekre vezethető vissza:

- A megnyitás: importálás egy üres dokumentumba. Nem látjuk, de pont olyan, mintha minden megnyitáskor begépelnénk az egész dokumentumot, csak a visszavonható műveletek listáját a végén töröljük, és a dokumentumot „nem módosított” állapotba tesszük.
- Másolás és beillesztés: lehetőség van részleges exportra, majd ennek eredményét a egy létező dokumentumba importáljuk.
- Mentés: exportálás egy olyan fájlnevet használva, ami már létezik. Ekkor persze odafigyel a felhasználói felület, hogy ne kérdezze meg: felül akarjuk-e írni a dokumentumot.

Ez a magyarázata annak, hogyha „belenézünk” egy ember számára is többé-kevésbé olvasható formátumba (Writer esetén gyakorlatilag csak a DOC nem ilyen), akkor ha azt egy másik program készítette, a Writerben egyetlen karaktert módosítva a fájlban mentés során az eredmény egész más is lehet. Persze a dokumentum modellje szemantikailag meg fog egyezni (többé-kevésbé), de a használt szintaxis egész más is lehet. Az adott formátum több nyelvi eleme is rendelődhet egy belső Writer koncepcióhoz, majd a mentés során ezt a belső reprezentációt következetesen csak egy adott nyelvi elem felhasználásával fogja az export szűrő kifejezni.

Időről időre emberek arra gondolnak, hogy a LibreOffice mögött álló „konverziós gépezetet” ki szeretnék emelni a szoftverből. A fentiek alapján láthatjuk, hogy ez miért lehetetlen: az import és export szűrők a belső dokumentummodellt használják közös nyelvként: tehát ha két formátum között konvertálni szeretnénk, szükségünk van az egész Writerre.

Ettől függetlenül nem kell elkeseredünk, ha ilyenre lenne szükség: tömeges konvertálásra mind parancssorból, mint a LibreOffice UNO API segítségével lehetőség van. Ez utóbbi a többszálú konvertálást is lehetővé teszi. Csak azt lehetetlen elkerülni, hogy a teljest szoftvert telepíteni kelljen.

Mielőtt megvizsgálnánk az egyes fájlformátumokat, még tekintsük át, hogyan tudjuk ezt a közös nyelvet, a dokumentummodellt megvizsgálni.

A dokumentummodell építőkockája a paragrafus. Egy lehetséges mód ezek vizsgálatára, ha az adott nyitott dokumentum tartalmáról egy XML dumpot készítünk:

```
SW_DEBUG=1 ./soffice.bin --writer
```

Majd ha a dokumentumot olyan állapotra hoztuk, amit vizsgálni szeretnénk, Shift-F12 hatására egy `nodex.xml` fájlba készül el a belső reprezentáció kiírása.

Egy másik, ehhez hasonló technika, a GDB hibakereső használata. Egy-egy C++ osztályhoz Python nyelven lehet a GDB számára pretty-printereket írni, amik segítik a hatékony megjelenítést. Ennek eredményeképpen, ha a `pDoc` egy `SwDoc*`, a következő parancs a hibakeresőben kiírja a dokumentum összes paragrafusát:

```
print pDoc->GetNodes()
```

Ez különösen akkor hasznos, ha a dokumentum modelljének változását sorról sorra szeretnénk követni.

Harmadrészt, a normál UNO API-t használva is megvizsgálhatjuk modellünket, ez különösen akkor hasznos, ha az adott szűrő amúgy is ezt az API-t fogja

használni munkája során. Basic makrót futtatva, a `ThisComponent` objektumon iterálva kapjuk meg a paragrafusokat, illetve egy-egy paragrafuson iterálva kapjuk meg az egyes szövettartományokat (amelyeken belül az egyes karakter-tulajdonságok is egyeznek).

3. Writer fájlformátumok: ODF, Word formátumok és a többiek

3.1. ODF szűrő

Tekintsük át, tipikusan milyen formátumokkal dolgozik a Writer, és ezek hol találhatók a forráskódon belül!

Az ODF szűrő a Writer saját formátuma, egyedül itt garantált az, hogy az import és export veszteségmentes. Persze hibák itt is előfordulhatnak, de ezek egyrészt ritkák, másrészt itt legalább az elméleti lehetőség megvan a veszteségmentes konverzióra, mivel a Writer belső reprezentációja igen közel áll az ODF jelöléséhez.

Például a paragrafusok UNO API-ban megjelenő tulajdonságai egy az egyben megfeleltethetők a használt XML elem egyes attribútumainak.

Az ODF import/export szűrő nagy része maga is az UNO API-t használja, így más szűrőknél mindig egy támpontot adhat a következő technika: megnézzük, az ODF szűrő hogyan kezel egy szituációt, vagy ha az adott szűrő a belső API-t használja, akkor megnézhetjük az UNO API idevágó implementációját.

A szűrő forráskódja nagyrészt az `xmloff/` könyvtár alatt található, valamint egy kisebb része az `sw/source/filter/xml/` alatt.

Az ODF szabványt nem csak szöveges formátumban, hanem Relax-NG sémában is karbantartják, ez alapján pedig automatikus validáció is végezhető. Ha nem vagyunk biztosak benne, hogy Writer hibával állunk szembe, vagy a dokumentum érvénytelen, használjunk egy ODF ellenőrzőt, például:

http://odf-validator2.rhcloud.com/odf-validator2/

3.2. RTF szűrő

Az RTF formátumot eredetileg olyan céllal találták ki, hogy egy olvasható HTML-jellegű formátumot kapjanak (szemben az akkor elterjedt egyéb bináris formátumokkal), ami azonban támogatja a Writer jellegű szövegfeldolgozók összes tulajdonságát. Gondoljunk arra, hogy a HTML nyelv nem foglalkozik az oldalmérettel, stb. Nyilván használhattak volna valami XML alapú megoldást, de 1987-ben még nem volt XML. :-)

A jelenlegi RTF export a LibreOffice 3.3-ban mutatkozott be, ez egy belső szűrő, a kód jelentős része megosztott a DOC és DOCX expoterrel.

Kódja az `sw/source/filter/ww8/rtf*` alatt található.

A jelenlegi RTF import a LibreOffice 3.5-ben jelent meg először, ez egy UNO szűrő, leginkább azért, mert így készült el a később bemutatott DOCX megnyitó is, és így az RTF és DOCX megnyitó egy része közös.

Ezeket a szűrőket nagyrészt a cikk szerzője követte el.

3.3. DOC szűrő

Valószínűleg a legrégebbi – jelenleg is használatban lévő – Writer szűrő. Ugyan a `binfilter` modulban található StarOffice 5.2 (és korábbi verziók) által használt szűrők még régebbiek voltak, ezek mára már csak a git tárolóban turkáló régészek számára hozzáférhetőek.

Mind az export és az import belső szűrő, és akkor itt álljunk is meg egy kicsit.

Mit jelent a belső és az UNO szűrő? A belső szűrő a Writer C++ API-ját használja közvetlenül, ami azt jelenti, hogy a dokumentummodell változtatása után mindig újra kell fordítani, valamint technikailag a Writer része. Hívják még natív szűrőnek is. Ezzel ellentétben az újabb szűrők már az UNO API-t használják: ez valamivel absztraktabb, és ennek következtében nem csak C++-ból használható. További előnye, hogy ez ritkán változik, minden kiadásban dokumentálásra kerül, ha bármilyen inkompatibilis változást hajtottak végre a fejlesztők, és ebből következően a Writer módosítások után tipikusan nem kell újrafordítani, tehát hatékonyabb fejlesztést tesz lehetővé.

Mondhatná azt valaki, hogy az absztrakciónak viszont mindig ára van: valamit elvesztünk vele. Jelen esetben ettől nem kell félni, az előző alfejezet említette, hogy a veszteségmentes ODF szűrők is ezeket az API-kat használják, tehát import/export szűrő szempontjából az UNO API is teljes hozzáférést biztosít. Mindezeket után érthetjük meg a viccet, miszerint technikailag a Writerben az ODF csak egy külső formátum, míg a DOC natívan támogatott. ;-)

Visszatérve a DOC formátumra, ez egy kereskedelmi formátum (nem egy független testület alakítja ki, nem lehet a fejlesztési irányba beleszólni, stb), de [MS-DOC] néven elérhető a specifikációja.

Mivel bináris formátumról van szó, a fájl struktúrái közvetlenül megfeleltethetőek C struktúráknak, ennek következtében az import/export kódja részben megosztott.

Akit elsőre megijeszt a bináris formátum, annak érdemes használnia az mso-dumpert:

http://cgit.freedesktop.org/libreoffice/contrib/mso-dumper/

Az ebben található doc-dump tool (szintén jelentés részben a szerző munkája) segíthet a DOC szűrőkkel kapcsolatos problémák megértésében.

3.4. DOCX szűrő

Mielőtt belemerülnénk a DOCX szűrő rejtelmeibe, tisztázzuk mi az a tokenizáló és a domain mapper. A tokenizálót tipikusan import szűrőkkel kapcsolatban szokták használni, lényege, hogy az adott formátumból kiolvassa a byte-okat és ezekből egy token-folyamot állít elő. XML esetén erre XML értelmezőként szoktunk hivatkozni, más formátumok esetén ennek nincs feltétlenül saját neve. A Writer szűrők esetén a tokenizáló még több feladatot is ellát: például a hiperhivatkozások leírására a DOCX szabvány két módot is biztosít: a tokenizáló feladata, hogy ezeket egységesítse, a szövegként leírt hexadecimális számokat, számokká alakítsa, és ehhez hasonló feladatokat lásson el.

A domain mapper feladata pedig, hogy az adott formátum koncepcióit (domainjét) a Writer koncepcióhoz rendelje (mappelje), például a DOCX szakaszait a legtöbb esetben a Writer oldalstílusainak lehet megfeleltetni, stb.

Míg a DOC filter esetén ez a két feladat összemosódott, a DOCX importer esetén ez határozottan szét lett választva, így vált lehetővé, hogy az RTF és a DOCX importer közös domain mappert használjon.

Visszatérve a DOCX formátumra, sajnos ennek is legalább két verziója van: az ECMA „szabvány”, valamint az a változat amit a Microsoft kezdetben implementált.

Nem kell nagy különbségekre gondolni: például a paragrafusok két szélső margójára az egyik leírásban left/right jelzővel kell hivatkozni, a másik esetben

start/end néven. Az importnak egyszerűen csak kezelni kell mindkettőt, az export esetén a felhasználó határozhatja meg, hogy melyik leírást szeretné. Mivel a legtöbb felhasználó a DOCX export eredményét Microsoft termékekkel akarja később megnyitni, az alapértelmezés a Microsoft-féle megvalósítás írása.

A DOCX importer a régebbi, még a Sun kezdte el készíteni, túlbonyolított kódja a `writerfilter/` alatt található. A tokenizáló XSLT nyelven készült, ez transzformálja a szabvány sémáját C++ kódba... :-)

4. Tesztek

Különféle teszt fajtákat készíthetünk, attól függően, hogy mi a célunk. A fájlformátum problémák esetén a a modell teszteket használjuk.

Import esetén a fájlt az import szűrő tölti be, majd az eredmény UNO modelljén értékelünk ki kifejezéseket.

Export esetén egy import → export → import láncon megy végig a dokumentum. Ennek az az előnye, hogy így a teszt dokumentumok továbbra is fájlokban tárolhatók (nem kódból kell felépíteni őket), valamint a kiértékelendő kifejezések is ugyanazon nyelven fogalmazhatóak meg, mint az import tesztek esetében.

5. Elérhetőségek

- LibreOffice honlap: https://www.libreoffice.org/
- A diák és ezen cikk elérhetősége: http://vmiklos.hu/odp/