# Tartalomjegyzék

# 1. fejezet

# Vajna Miklós: Hogyan készítsünk Writer funkciókat?

## 1.1. Bevezetés

Nem csak a funkciók számítanak. Mielőtt belevetnénk magunkat a funkciók készítésének rejtelmeibe, ne felejtsük el, hogy nem ez az egyetlen módja a LibreOffice projekt segítésének. Segíthetjük a felhasználókat (levelező lista, fórum, IRC), tevékenykedhetünk a minőség javításán (hibajelentések készítése, nem megerősített hibák reprodukálása), útmutatókat írhatunk, a design csoport az inkább csak a fejlesztéshez értő fejlesztőknek ad tanácsot jó felhasználói felületek kialakításához. Az infrastruktúrát üzemeltető önkéntesek nélkül a többiek nem tudnának tevékenykedni.

De még egy fejlesztőre váró kihívások közül is csak egy az új funkciók készítése: sok időt visz el a hibák javítása, tesztelés, a létező kód átrendezése, új önkéntesek segítése, dokumentálás. Ezek mind legalább annyira fontosak, mint az új funkciók fejlesztése. Ennek ellenére, ez a cikk a továbbiakban kizárólag ezzel a témával foglalkozik.

Mielőtt továbbhaladnánk, nézzünk két példát új funkciókra, amik a LibreOffice 4.0-ban fognak bemutatkozni:

- Korábban csak egy adott ponthoz fűzhettünk megjegyzéseket. Az új *hozzászólás szövegtartományhoz* funkció lehetővé teszi, hogy – egy kijelölés után – olyan megjegyzést készítsünk, mely eltérő kezdő és végpozícióval rendelkezik.
- A *más fejléc/lábléc az első oldalon* funkció lehetővé teszi, hogy egyetlen oldalstílust használva ne csak a jobb/bal oldalon lehessen külön fejléc/lábléc, hanem az első oldalon is.
-

Mindkét funkciót régóta kérik a felhasználók – ezt tekinthetjük általános szabálynak is: egy, a 80-as évek óta fejlesztett irodai szoftvercsomagba érdemes

meggondolni, milyen új funkcióra van szükség, csak akkor érdemes nekikezdeni valami újnak ha arra valós igény van.

Mindezek után tekintsük át azt a nyolc lépést, amit minden új funkció elkészítése során végig érdemes követnünk:

1. dokumentum modell
2. UNO API
3. megjelenítés (layout)
4. szűrők
5. felhasználói felület
6. tesztek
7. dokumentáció
8. specifikáció

A cikk hátralevő részében ezeket a lépéseket részletezzük.

## 1.2. Dokumentum modell

Azt lehetne gondolni, hogy egy funkció fejlesztése során csak ezzel kell foglalkozni. A Writer klasszikus Modell-View-Controller paradigmában gondolkozik. Egy dokumentum egy `SwDoc` osztálynak feleltethető meg. Ezen belül az építőkockánk a paragrafusok (`SwNode` osztály). A forráskódban ezek az `sw/source/core/` könyvtár alatt találhatóak.

Érdekesség, hogy nincs külön szöveges csomópont a szövegtartományoknak[1], helyette egy `SwpHints` osztályba gyűjtve minden paragrafuson belül változó tulajdonsághoz egy `SwTxtAttr` tartozik, kezdeti pozíció, végpozíció és érték információkkal.

Ezen kívül még sok a kivétel, külön kollekcióban (nem paragrafusokhoz csatolva) találhatók meg például az oldalstílusok (`SwPageDesc` osztály). Az „első oldali fejléc” funkcióhoz például ehhez az osztályhoz kellett az `aMaster` és `aLeft` tagok mellé egy `aFirst` tagot is felvenni.

Aki szeret példákon keresztül tanulni, annak érdemes az

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

opcióval indítania a Writert, ilyenkor a Shift-F12 a `nodes.xml` file-ba kiírja a jelenleg aktuális dokumentummodellt.

Másik lehetőség, ha UNO-n keresztül, makróból szeretnénk végigiterálni a paragrafusokon:

```
enum = ThisComponent.Text.createEnumeration
para = enum.NextElement
para = enum.NextElement
xray para
```

Így egy tetszőleges paragrafus tulajdonságait tekinthetjük meg.

[1] Olyan szakasz egy paragrafuson belül, ahol a karakterek tulajdonságai is – pl. betűtípus – megegyeznek.

## 1.3. UNO API

Makrókból nem férhetünk hozzá közvetlenül a Writer belső dokumentum modelljéhez, ehhez annak UNO API-ját kell használnunk. Ezért fontos, hogy az új funkciónkat elérhetővé tegyük ezen a publikus API-n keresztül is.

Ez egyébként is hasznos, mivel ha makróból már tudjuk írni/olvasni az új funkciót, akkor a felhasználói felület elkészítése előtt is már tudjuk azt tesztelni. Ezen kívül az UNO alapú (ld. később) szűrők csak az UNO API-n elérhető funkciókat tudják menteni, valamint a tesztesetek is az UNO API-t használják legtöbbször. Az Writer UNO API kódját az `sw/source/core/unocore/` alatt találjuk.

Nézzünk példákat! Ha a környezetérzékeny térközöket szeretnénk bekapcsolni egy paragrafusra:

```
enum = ThisComponent.Text.createEnumeration
para = enum.NextElement
xray para.ParaContextMargin
para.ParaContextMargin = True
```

Ha szeretnénk, hogy az első oldalon más fejléc/lábléc legyen:

```
oDefault = ThisComponent.StyleFamilies.PageStyles.Default
oDefault.HeaderIsOn = True
oDefault.HeaderIsShared = False
oDefault.FirstIsShared = False
```

Ha megjegyzést szeretnénk csatolni egy szövegtartományhoz:

```
oTextField =
oDoc.createInstance("com.sun.star.text.TextField.Annotation")
oTextField.TextRange.String = "Tartalom"
oDoc.Text.insertTextContent(oCurs, oTextField, True)
```

## 1.4. Layout

A layout célja megjeleníteni azt, ami eddig csak a dokumentum modelljében volt elrejtve, azaz a „View” az MVC-ből. A Writer esetében ennek is saját dokumentum modellje van, aminek az építőkövei a keretek. Egy megnyitott dokumentum egy `SwRootFrm`-nek felel meg, egy oldal egy `SwPageFrm`-nek, majd ezen belül egy paragrafus egy `SwTxtFrm`-nek. Egy modell elem több layout elemnek is megfelelhet, gondoljunk csak arra az esetre ha egy fejléc több oldalon is megjelenik, vagy egy paragrafus átfolyik a következő oldalra. Ehhez az `1:N` megfeleltetéshez a Writer a az `SwClient` / `SwModify` mechanizmust használja: egy kliens csak egy kiszolgálóhoz lehet regisztrálva, viszont a kiszolgálóknak lehet több kliense, melyeken az `SwClientIter` osztállyal lehet iterálni.

Ha példát szeretnénk látni a layout felépítésére, a dokumentum modelljéhez hasonló módon megtehetjük, csak F12-re lesz szükségünk a Shift-F12 helyett.

## 1.5. Szűrők

Próbáljuk meg túlélni az újraindítást! Ehhez a szűrők úgy segítenek bennünket, hogy az exporterek egy `SwDoc` példányt mentenek egy streamre, az importerek pedig egy stream alapján felépítenek egy (eredetileg üres) `SwDoc`-ot.

Alapból minden szűrő „alien", ami azt jelenti, hogy ismert, hogy információt vesztenek a konvertálás során. Egyetlen kivétel az ODF szűrő, amire emiatt „own"-ként hivatkoznak. Ennél a szűrőnél garantált a veszteségmentesség, ezért is kell kiterjeszteni *minden* új funkció implementálásakor.

A szűrők lehetnek UNO-alapúak (pl. DOCX import, RTF import), lehetnek beépítettek (belső sw API-t használják, pl. DOC import, DOC/DOCX/RTF export), végül lehetnek kevertek, mint amilyen az ODF szűrő (főleg UNO, de kis részben belső). Ezek forráskódjai az `sw/source/filter/`, `writerfilter/`, ill. az `xmloff/` alatt találhatóak.

## 1.6. Felhasználói felület

Eljutottunk oda, hogy a felhasználók számára is élvezhetővé tudjuk tenni a munkánkat. A felhasználói felület a színfalak mögött lehet régi vagy új.

A jelenlegi ablakok nagy része még a régi formátumot használja, tervezői vélhetőleg „ne használjunk egeret az UI elkészítéséhez!" felkiáltással tervezhették. Tipikusan egy `.src`, `.hrc` és egy `.cxx` file készült minden ablakhoz. Minden elem rögzített pozícióval / mérettel rendelkezett. Például:

```
CheckBox CB_SHARED_FIRST
{
HelpID ="svx:CheckBox:RID_SVXPAGE_HEADER:CB_SHARED_FIRST";
Pos = MAP_APPFONT ( 12 , 46 ) ;
Size = MAP_APPFONT ( 152 , 10 ) ;
Text [ en-US ] = "Same content on first page" ;
};
```

Számos probléma van ezzel a megoldással, például ha új elemet adunk az ablak közepéhez, az összes többi vezérlőt le kell mozgatni kézzel, vagy a gomb szélessége a leghosszabb fordítást kell figyelembe vegye, stb.

Ezzel szemben sokkal előremutatóbb Caolán új Glade-alapú megoldása[2], bár ha csak egy-egy új vezérlőt adunk létező ablakokhoz, egyelőre ritkán találkozunk velük.

Az UI gyakran közös, nem lehet alkalmazás-specifikus, így az UI és a dokumentum modell közötti interakcióra a feltalált megoldás az `SfxItemSet` lett. Ha egy jelölőnégyzetet szeretnénk `SfxItemSet`-be tenni:

```
aSet.Put(SfxBoolItem(nWSharedFirst,
    aCntSharedFirstBox.IsChecked()));
```

Ez után kell az `SfxItemSet`-ből a dokumentum modellbe (jelen esetben `SwPageDesc`) rakni:

```
rPageDesc.ChgFirstShare(((const SfxBoolItem&)
    rHeaderSet.Get(SID_ATTR_PAGE_SHARED_FIRST)).GetValue());
```

[2] `http://conference.libreoffice.org/talks/`

Ellenkező irányban is hasonló megoldásra lesz szükségünk. Erre számos létező példa található: a közös kód az `svx/source/dialog/` és a `cui/` alatt, a Writer-specifikus kód az `sw/source/ui/` alatt.

## 1.7. Tesztesetek

Miért ismételnénk meg régi hibákat, ha választhatunk újak közül is? :-) A tesztesetek ebben segítenek minket.

A LibreOffice-ban unitcheckeket (*minden* részleges modul fordítás végén lefutnak), slowcheckeket (minden teljes fordítás végén futnak), valamint subsequentcheckeket (külön `make subsequentcheck`-re futnak csak) használunk. Új funkciók esetén legtöbbször egy új slowcheck-et készítünk: ez hozzáférést ad az UNO API-hoz, viszont minden fejlesztő gépén lefut, így széles tesztelést biztosít.

Tesztelés során legtöbbször a dokumentum modellt (UNO API) vagy a layoutot (XML kiírás + XPath kifejezés kiértékelése) vizsgáljuk. Ez utóbbira egy példa:

```
CPPUNIT_ASSERT_EQUAL(OUString("Első fejléc"),
    parseDump("/root/page[1]/header/txt/text()"));
```

Import / export tesztek esetén először minimális reprodukáló dokumentumot készítünk. A tesztelés során vagy importálunk, vagy egy import-export-import szekvenciát hajtunk végre, attól függően, hogy mit akarunk tesztelni. Ez azért jó, mert így a tesztelendő dokumentumot mindig file-ban tárolhatjuk (nem kódból kell felépíteni), valamint a tesztelést is egy létező `SwDoc` példányon tudjuk elvégezni, nem pedig egy valamilyen formátumban elmentett file-t kell vizsgáljunk.

Az elkészült (importált) dokumentum modelljét az UNO API-val könnyen vizsgálhatjuk. Pl. ha valamilyen nem-ASCII karakter probléma volt, sokszor elég csak a dokumentum hosszát (karaktereinek) számát vizsgálni:

```
CPPUNIT_ASSERT_EQUAL(6, getLength());
```

A teszt megírása után a teszt futtatása következik. Ha sikeres, érdemes visszavonni a javítást vagy funkciót, és megbizonyosodni arról, hogy a teszt hibák jelez, vagy is a teszt jó. Ha ez stimmel, eldobhatjuk a visszavonást és összevonhatjuk a két commitot, így később nem kell keresni, hogy melyik módosítást tesztelte a teszteset.

A Writer tesztjeit az `sw/qa/` alatt keressük.

## 1.8. Dokumentáció

A felhasználói dokumentáció a súgó, ami F1-re jelenik meg. Új tartalmat ehhez úgy érdemes hozzáadni, hogy a felhasználói felületen kikeressük a jövendőbeli szakasz szomszédos részeit, majd `git grep`-pel rákeresünk a létező tartalmakra a `helpcontent2/` alatt.

Fontos, hogy míg normál kód módosításkor elég a fordítás a változtatásunk teszteléséhez (a `linkoo`-nak köszönhetően), így fordítás után egy telepítés is kell (`make dev-install`) mielőtt kipróbáljuk annak eredményét. További trükk,

hogy minden paragrafusnak kell egy egyedi azonosító a fordításokat segítendő, de ez csak egy file-on belül kell egyedi legyen, tehát egyszerűen másoljunk le egy létezőt, majd tetszőleges algoritmust választva tegyük egyedivé[3].

A fejlesztői dokumentációra mindössze annyi a követelmény, hogy minden új osztálynak legyen legalább egy egysoros `doxygen` leírása, hogy mit csinál – preferáltan minden publikus metódusra is, de ez már nem követelmény.

## 1.9. Specifikáció

Azért nem ezzel kezdünk egy új funkciót, mert mindent ODF-be mentünk, ami egy nyílt szabvány, és csak az kerül bele az ODF-be, amit már 3 független implementáció életképesnek mutat. Ennek megfelelően alapból a LibreOffice egy *ODF 1.2 Extended* formátumba ment, ami tartalmazza a saját kiegészítéseinket, majd ezeket javaslatként eljuttatjuk az OASIS-hoz.

Egy ilyen módosítási kérelemnek tartalmaznia kell, hogy mi a motiváció a változtatásra, a szabvány szövegéhez valamint a RELAX NG sémához milyen változtatásokat szeretnénk eszközölni, végül részleteznie kell a változtatás várt hatását (visszafele kompatibilitás kezelése, stb).

## 1.10. Elérhetőségek

A szerző ezúton is elnézést kér, hogy sok – a cikkben szereplő – témát csak érintőlegesen említett, csak a Writer belső működéséről vastag könyvet lehetne írni, a cél leginkább a figyelemfelkeltés volt. Az alábbi linkek további kérdések esetén remélhetőleg segítséget nyújtanak.

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

[3] Egyszerű inkrementálás, ínyenceknek kvadratikus próba tökéletesen megfelel. :-)