Az xc2 fájlformátum család

# Bevezető

Gyakran használok XML fájlformátumot, ugyanis a DOM könnyedén áttekinthető segítségével, viszont rengeteg, egy értelmező program számára felesleges adatot szállít egy ilyen állomány. Ezért döntöttem úgy, hogy bizonyos kompromisszumok, megkötések bevezetésével csökkenteném az adatállományok méretét úgy, hogy megmaradjon karakteres felépítésük, a schema állomány kíséretében a DOM továbbra is könnyedén olvasható és értelmezhető maradjon.

# Főbb jellemzők

Az XML formátumban a leggyakoribb karakterek közé tartozik a <, > és a / jelek. Ezek adják meg a dokumentumok jellegzetes kinézetét és ezek jelzik a fordítónak hol kezdődnek és érnek véget a kulcsszavak, tag-ek.

Ezek 3 fajta jelölést alkotnak:

* Nyitó tag: <azonosító>
* Záró tag: </azonosító>
* Üres objektum: <azonosító />

A fájlméret csökkentéséhez – úgy, hogy a karakteres felépítés megmaradjon – elengedhetetlen, hogy ezektől megszabaduljunk, ehhez bevezettem egy megkötést, a tag-eknek case-sensitiv-nek kell lennie!

A 3 fajta jelölés a következőképpen módosult:

* Nyitó tag: AZONOSÍTÓ (UPPERCASE)
* Záró tag: azonosító (lowercase)
* Üres objektum: aZonosító (camelCase)

Eredetileg szabadszövegesek az azonosítók, így kvázi végtelen különböző objektum típus létrehozható belőlük. Ha ezt limitáljuk 216-on darabra, azaz 65536 lehetőségre már az értelmező program is egyszerűbbé válik és a fájlméret is hatalmasat csökken. Így 16 biten, azaz 2 bájton tárolható egy típus jelölés. Innen is az xc2 megnevezés: Xml Compressed into 2 bytes. Bár nem kötelező, viszont a szöveges olvashatósághoz ajánlott limitálni a szimbólumokat a megjelníthető karakterekre, azokon belül is olyanokra, amelyeken a case-sensitivítás értelmezhető, így az angol ábécé betűi maradtak. Ez két bájton 26\*26 = 676 lehetőség. Az értelmezőn tovább egyszerűsíthetünk, ha tiltjuk a 16-os számrendszerben használt A..F szimbólumokat is, így 20\*20 = 400 lehetőség maradt. A tárolandó dokumentum bonyolultságától függően választhatjuk a 676 szimbólumos vagy a 400 szimbólumos megoldást is.

A fenti számítások és lehetőségek figyelembevételével és azok számának csökkentése nélkül fenntartottnak nyilvánítottam a következő karakter párosokat (nem hozható létre üres objektum):

* Xc2 dokumentum típus: X!-x!
* Xc2 séma fájl: X$-x$
* Dokumentum header: X@-x@
* Dokumentum cím: X+-x+
* Dokumentum body: X&-x&
* Szöveges megjegyzések: X# -x#

# Kiterjesztések

## Az .x2d kiterjesztés

A dokumentumok számára fenntartott kiterjesztés, a fejrészben jelölendő, hogy 676 vagy 400 szimbólumot használó verzióról van szó, ezekhez az X6 és az X4 karakterpárosok állnak rendelkezésre, záró páros nélkül! Az X0 jelzi, hogy bármilyen adat előfordulhat, nem csak karakterekből összeállított, azaz mind a 65536 lehetőség rendelkezésre áll, ilyenkor fel kell készíteni az értelmezőt arra, hogy a fenntartott karakterpárosok esetleg előfordulhatnak az adatok közt is, továbbá a fenntartott párosokat letiltani a séma állományban, ha nem akarjuk őket felüldefiniálni.

Tartalma a következőképpen alakul:

1. X!
2. X@
3. X6
4. X$”Séma fájl útvonala.”x$
5. X+”Dokumentum címe.”x+
6. x@
7. X&
8. X#”Ez itt a dokumentum teste.”x#
9. x&
10. x!

WhiteSpacek elhagyásával a következő kódot kapjuk:

1. X!X@X6X$”Séma fájl útvonala.”x$X+”Dokumentum címe.”x+x@X&X#”Ez itt a dokumentum teste.”x#x&x!

A fenti kódrészlet az alábbi HTML jellegű kódként értelmezhető:

1. <!DOCTYPE ”Séma fájl útvonala.”>
2. <XML>
3. <HEAD symbols=676>
4. <TITLE>
5. Dokumentum címe.
6. </TITLE>
7. </HEAD>
8. <BODY>
9. <!--Ez itt a dokumentum teste.-->
10. </BODY>
11. </XML>

Megfigyelhető, hogy megspóroltunk 145 – 93 = 52 karaktert, ami a 52 / 145 = 35,86%-a a dokumentum méretének, tehát több, mint az egyharmadával csökkent.

## Az .x2s kiterjesztés

Séma állomány, felépítése még kidolgozás alatt áll.

## Az .x2h kiterjesztés

Huffman-kódolással történő tömörítést tesz lehetővé, ha az xc2 által bevezetett rövidítések mellett is még mindig nagyobbak a fájlok, mint az elvárt. Azért javaslok Entrópia alapú kódolást, mert, ahogy az eredeti XML formátumban igen gyakoriak a < és > jelek, addig az xc2-ben az X és az x karakterekből lesz több, mint a többiből, így van értelme gyakoriság szerint rendezni a karaktereket. Ha főképp szövegből áll a dokumentum, akkor egy, a Morse kódhoz hasonló bináris fát kapunk eredményül, csak épp előfordulhat, hogy az X karakterek ábrázolásához nem kell 4 bit sem, továbbá különbség, hogy csak a levél csomópontokra kerülnek karakterek. Az értelmezéshez szükséges fát a fájl elején kell elhelyezni, első bájtra a gyökér csomópontot, majd a fájl megfelelő bájtjára mutató 16bit hosszú pointer, hogy hol kezdődik a bal és hol a jobb gyermekcsomópont. Egy csomópont felépítése: 1 bájt a tárolt adat, ha nincs, akkor értéke unsigned 0; 2 bájt bal pointer, 2 bájt jobb pointer. Ha a csomópont levél, tehát tartalmaz karaktert, akkor a következő bájton nem pointer lesz, hanem a következő eltárolt csomópont adatai. A fa eltárolása után 0xffff értékű két bájt jelzi, hogy véget ért a fejrész. A következő bájttal már az .x2d állomány következik tömörítve, a tömörített adatcsomag végével a fájl is véget ér.

Az .x2s kiterjesztésnek nincsen tömörített verziója, ugyanis eleve nem képes túl nagyra nőni az állomány, hiszen nem fordulhatnak elő benne nagyobb mennyiségű ismétlődések, értem ez alatt, hogy egy objektumból nem lesz több száz példány, hiszen csak a típusának definícióját tartalmazza.

# Adattípusok

## Szöveg

Egy " karakter jelzi a kezdetét és a végét. A két idézőjel közt megtalálható bájtsorozatból bármely szabadon választott karakterpáros nem értelmezhető utasításként, kizárólag adatként! Így a "X!x!X!x!X!x!” nem dokumentumok sorozatát jelenti, hanem csak egy sorminta.

A fenti szabály alól sem jelent kivételt a sortörést jelző X- karakterpáros, ilyenkor a szöveget idézőjellel le kell zárni, majd az X- használata után ismét meg kell nyitni. Ha mégis a szövegen belül szeretnénk sortörést elhelyezni, akkor szabadon használható a hagyományos \n vagy <BR> karakterláncok használata, ugyanis ezek nem tartoznak egy xc2 értelmező hatáskörébe.

## Számok

Kizárólag az egész számokat értelmezzük számként az xc2 fájlokban, azokat is 16-os számrendszerben, azaz HEX-ben kódolva, de karakteresen szállítjuk. Ezek kezdetét az x0 vagy az x- jelzi, előbbi a pozitív, utóbbi a negatív számokat. Lezáró párjuk nincsen. Ha az értelmező lehetővé teszi, akkor használható a hagyományos 0x jelölés az unsigned számokra.

Törteket, komplex számokat és mindent, ami az egész számok halmazán kívül esik szövegként vagy felhasználó által definiált objektumokban szállíthatunk.

Minden számot hosszától függetlenül jelölni kell, tehát az x010x020 két egymástól független számot, azaz a 16-ot és a 32-őt jelenti, így két adatot jelent, míg az x01020 viszont egy darab adat, egy szám, tehát 4128-at jelent. Ebből következik az is, hogy az x030x-40 értelmezhető, két számot jelent, míg az x030-40 hibás, így nem értelmezhető egy számként sem.

## Műveletek

Az xc2 adatok már kiszámolt vagy teljesen nyers formájú tárolására és továbbítására szolgál, így műveletek nem végezhetőek a dokumentumon belül. A felhasználó szabadon létrehozhat műveletvégzésre szolgáló karakterpárosokat vagy olyan objektumokat, amiket értelmezés után, a belőlük kigyűjtött adatok szerint úgy dolgoz fel, hogy azok közt műveleteket végez, de ezek már nem az xc2 hatáskörébe tartoznak.

# Előre definiált dokumentum sémák

Előre definiált esetről akkor beszélhetünk, ha a telepített értelmező rendelkezik séma állománnyal, így a dokumentum olvasása közben a saját direktívái alapján dolgozza fel a kapott adatokat, ezt a dokumentum fejrészében azonosítóval szükséges jelölni.

A dokumentum fejrészében az X$ séma jelölő, ha nem szöveget szállít, hanem egy számot, akkor az annak a számnak megfelelő azonosítójú előre definiált séma állomány szerint kell a dokumentumot értelmezni.

Példa: `X!X@X6X$x010x$X+”Cím”x+x@X&X#”Dokumentum teste.”x#x&x!` dokumentumhoz az értelmező a 16-os azonosítójú séma állományt fogja megnyitni.

Könnyíthetünk a felhasználók dolgán, ha halmozott kiterjesztést használunk és az utolsó pont elé bele írjuk a fájlnévbe az előre definiált séma azonosítóját is. Például: Adatok.x010.x2d, így az értelmezőnek sem szükséges a fájlban keresni az azonosítót, viszont feldolgozás előtt érdemes ellenőrizni, hogy a fájlnévben említett sémaazonosító megegyezik-e a fájlban hivatkozott azonosítóval és természetesen a fájl feldolgozása előtt a validálás sem maradhat el.

## Az x010 azonosítójú séma

Az x010-es azonosítót lefoglaltam egy projektemhez, fényvezérlő kódokat tárolni a segítségével. Mikor ezt a sémát terveztem, még nem készültek el a séma állományok definíciói, így ezt egy kifejezetten erre a sémára tervezett értelmező dolgozza fel, mely az Xc2S10\_Interpreter nevet viseli.

Az alábbi XML kódhoz nem tartozik sem DTD, sem XSD állomány, így az X2D-re fordított mása sem fog tartalmazni séma leírókat.

1. <XML>
2. <HEAD symbols=676>
3. <TITLE>
4. Chauvet 4play példa show
5. </TITLE>
6. </HEAD>
7. <CueList>
8. <KeyFrame>
9. <BytePair bl=1 br=92 />
10. <BytePair bl=2 br=92 />
11. </KeyFrame>
12. <KeyFrame />
13. <KeyFrame />
14. <KeyFrame />
16. <KeyFrame>
17. <BytePair bl=1 br=0 />
18. <BytePair bl=2 br=0 />
19. <BytePair bl=3 br=92 />
20. <BytePair bl=4 br=92 />
21. </KeyFrame>
22. <KeyFrame />
23. <KeyFrame />
24. <KeyFrame />
26. <KeyFrame>
27. <BytePair bl=3 br=0 />
28. <BytePair bl=4 br=0 />
29. </KeyFrame>
30. <KeyFrame />
31. <KeyFrame />
32. <KeyFrame />
33. </CueList>
34. </XML>

Megfigyelhető, hogy a fájlban a BODY tag el lett hagyva, helyette máris a CueList felirat szerepel, jelezve ezzel a dokumentum törzsének a feladatát.

Ebből az alábbi xc2 kódot készítettem:

1. X!
2. X@X6
3. X+”Chauvet 4play példa show”x+
4. x@
5. CL
6. KF
7. bP0x010x5C
8. bP0x020x5C
9. kf
10. kFkFkF
11. KF
12. bP0x010x00
13. bP0x020x00
14. bP0x030x5C
15. bP0x040x5C
16. kf
17. kFkFkF
18. KF
19. bP0x030x00
20. bP0x040x00
21. kf
22. kFkFkF
23. cl
24. x!

Az XML kód 446 karakter, míg az xc2 mindössze csak 154 és ugyan azt a dokumentumot jelentik. Így megspóroltuk a dokumentum hosszának több, mint 65%-át!

Ez a séma az alábbi objektumokat használja:

* CL és cl = <CueList>
  + KeyFrame-kből álló listát tárol.
* KF, kf és kF = <KeyFrame>
  + Egy idő pillanat a fényprogramban, a megváltozó adatokat listázza az új értékeikkel.
  + ꓱ üres objektum, ilyenkor nem változik semmi.
* bP = <BytePair />
  + Kizárólag üres objektum ꓱ.
  + Pontosan két számot hordoz paraméterként, amiket majd bájtként kell értelmezni.
* wB = <WaitBeatSignal>
  + Megakasztja a lejátszást, a következő ütemet jelző jelre folytatódik.
  + Kizárólag üres objektum ꓱ.
  + Nincs paramétere.
* wM = <WaitMilliSeconds>
  + Kizárólag üres objektum ꓱ.
  + 1 szám típusú paramétere van, megakasztja a lejátszást ennyi ms-re.
* FT = <FadeTime>
  + Kizárólag üres objektum ꓱ.
  + 1 szám típusú paramétere van. A következő értékig egyesével lépked el, köztük annyi várakozással, hogy végülis a paraméterben jelzett, ms-ben megadott késés jöjjön ki összegül
* FI és fl = <ForIteration>
  + A kezdő és záró karakterpáros közt szereplő utasításokat megismétli a paraméterében jelölt mennyiségben
* II és ii = <InfiniteIteration>
  + A kezdő és záró karakterpáros közt szereplő utasításokat ismételgeti leállításig.
  + Több objektum nem szerepelhet utána, csak a lista lezárása.
  + Szerepelhet benne több FI is.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -- | L | F | P | B | M | T | I |
| Alapállapot | >alap | >alap | >alap | >alap | >alap | >alap | >I |
| C |  |  |  |  |  |  |  |
| K |  |  |  |  |  |  |  |
| B |  |  |  |  |  |  |  |
| W |  |  |  |  |  |  |  |
| F |  |  |  |  |  |  |  |
| I |  |  |  |  |  |  |  |