Szövegfeldolgozás és XML

# Tananyag

Megismerkedünk az XML-el, a SAX feldolgozással és létrehozzuk egy saját implementációt egy keretprogramba beledolgozva.

## XML

Az XML rövidítés az eXtensible Markup Language (Kiterjeszthető Jelölő nyelv)-ből jön. Szöveges fájlban tároljuk az adatokat, címkékből (tag), attribútumokból és a tartalomból áll. Hierarchikus szerkezetű. SGML-re építve kezdték 1996-ban kidolgozni. 1998-ban jelent meg az ajánlás első változata. Az 1.0 verzió a legelterjedtebb, amit használunk.

### Célok a létrehozáskor

* Közvetlenül használható az Interneten
* Alkalmazások támogassák
* Egyszerű kezelhetőség
* Ember által olvasható és érthető
* Egyszerűen elkészíthető
* Dokumentum típus definíciók formális, tömör, gyorsan elkészíthető
* Szabadon használható és terjeszthető

### Szintaxis

#### Példa

<?xml version="1.0"?>

<!-- This is a comment -->

<example>

<teacher executive="true" filled="false"/>

<teacher executive="true">

<etrid>ZSLABCD</etrid>

<name>Zsako, Laszlo</name>

<email>zsako@ludens.elte.hu</email>

</teacher>

<teacher>

<etrid>MELEAET</etrid>

<name>Menyhart, Laszlo</name>

<email>menyhart@inf.elte.hu</email>

</teacher>

</example>

Az első sorban szerepel az XML deklaráció, mely megmutatja a használt XML verziót és esetleg a karakterkódolást, ami most hiányzik.

A második sorban egy megjegyzés szerepel.

A harmadik sorban a gyökérelem, az "<example>" található.

Majd egy üres elem (element) következik, ahol már két attribútum is található, az "<teacher>".

A következő sorban egy nyitó tag látható egy attribútummal.

Ezután 3 sorban szöveget tartalmazó csomópontok szerepelnek („etrid”, „name”, „email”).

A kilencedik sorban a „teacher” záró tag-je van.

Ezt újra egy teljes „teacher” node követi.

Végül a gyökérelem („example”) lezárásával végződik a dokumentum.

#### Nyitó tag-ek

Nyitó tag-ben az element neve kell, hogy legyen és az attribútumok szerepelhetnek, ha vannak.

#### Záró tag-ek

A HTML-vel ellentétben itt kötelező megadni a záró tag-eket. A nyitó tag nem szerepelhet anélkül.

A nyitó és záró tag-eknek megegyezőknek kell lenniük a betűket tekintve. Fontos, hogy a kis és nagy betűk különbözőeknek számítanak.

Az egymásba ágyazás nagyon fontos. Míg a HTML-ben nem fontos a záró tag-ek sorrendje, az XML dokumentum felépítése a leszármazásra épül, ezért itt a második a helyes.

<b>**<i>**Vastag és dőlt szöveg. HIBÁS!**</b>**</i>

<b>**<i>**Vastag és dőlt szöveg. HELYES!**</i>**</b>

#### Egy gyökérelem

Minden XML dokumentumnak egy gyökéreleme kell, hogy legyen. Ennek lehet gyereke, mely ugyancsak tartalmazhat egy leszármazottat.

<gyoker>

<gyerek>

<leszarmazott>...</leszarmazott>

</gyerek>

</gyoker>

#### Attributumok

Az attribútumok név-érték párként szerepelnek. Kötelező betartani, hogy az érték idézőjelek között van. Így csak a második példa helyes.

<oldal azon=0> - HIBÁS

<oldal azon="0"> - HELYES

#### Megjegyzés

A megjegyzést ugyanúgy kell írni, mint a HTML-ben.

<!-- Ez egy megjegyzés. -->

### Well-formed, jól formázott dokumentum

Azt az XML dokumentumot nevezhetjük jól formázottnak, aminek az elkészítésekor betartottuk a fenti szabályokat.

### Valid, érvényes dokumentum

Az érvényes (Valid) XML dokumentum jól formázott és ezen kívül a struktúrája megfelel a szerző által felállított szabályoknak. Ezeket a szabályokat is le kell írnunk, erre van a következő két lehetőség.

#### DTD

A DTD definiálja az XML dokumentum elemeinek struktúráját. Azt, hogy milyen elemekből állhat, azok milyen struktúrában épülnek fel. A DTD egy speciális definíció, melyet az XML dokumentum is tartalmazhat (belső deklarációként <!DOCTYPE …> node-ként) vagy egy külső fájlra is lehet hivatkozni. Csak az element-ek és az attribútumok neveit, előfordulásaik helyét, számát, sorrendjét lehet meghatározni ezzel.

#### XSD

Az XML Schema Definition vagy XSD a DTD-nek egy kiterjesztése. Az XML dokumentum struktúrája, adatainak típusa írható le benne XML formátumban. Definiálhatjuk benne az XML dokumentumban szereplő elemeket, attribútumokat, leszármazásokat, az elemek számára írhatunk előírásokat, adattípusokat használhatunk a megadás során és a rögzített adatokat is megadhatjuk. A DTD-nél jobban használható, több információt adhatunk meg. Támogatja az adattípusokat és a névtereket. Így nem csak szintaktikai, hanem szemantikai ellenőrzésre is lehet használni.

### Speciális karakterek

Vannak olyan karakterek, melyek feldolgozás megkönnyítése érdekében speciális tulajdonságokkal rendelkeznek. Például az "<" karakter a tag-ek kezdetét jelenti, így helyett a "&lt;" karaktersorozatot kell írni. Több ilyen karakter van, melyek helyett a helyettesítésükre definiált ENTITY karaktereket lehet írni. Az egyedek mindig a "&" karakterrel kezdődnek és a ";"-vel végződnek.

|  |  |  |
| --- | --- | --- |
| **Entity** | **Helyettesített jel** | **Leírás** |
| &lt; | < | kisebb, mint (less than) |
| &gt; | > | nagyobb, mint (greater than) |
| &amp; | & | és (ampersand) |
| &apos; | ’ | aposztróf (apostrophe) |
| &quot; | ” | macskaköröm (quotation mark) |
| &#337; | ő | a karakterkódok szám értékei is használhatóak |

### CDATA szekció

Olyan adathalmazokat is meg kell jeleníteni, ahol nehéz lenne megoldani a karaktercseréket vagy nem akarjuk pl. „K&H Bank” nevének a tárolásakor. Itt megadhatjuk, hogy az adott rész ne kerüljön feldolgozásra. Létre kell hoznunk egy CDATA, karakteradat szekciót. Így ebbe a részbe bármilyen adatot lehet írni. Kivétel a záró jelsorozatot nem lehet beírni a közepére, hiszen akkor ott érzékelné az adatcsomag végét.

<![CDATA[

A kezdő jelsorozat van az előző sorban. Akármilyen, nem feldolgozott szöveg következhet még „<” jel és „&” is. A végét jelentő karaktersorozat:

]]>

### Tag és element vagy node

Az element (elem) másik elnevezése a node ((fa) csomópont), ami a DOM-ból (lásd később) származik.

Egy element nyitó tag-ből, záró tag-ből és az element tartalmából áll. A nyitó tag a „<” (kisebb) jel, az elem neve, az esetleges attribútumok felsorolásából (név=”érték” formában) és „>” (nagyobb) jelből áll. A záró tag a „<” (kisebb) jel, „/” (per) jel, az elem nevéből áll. A TARTALOM lehet szöveg, vagy leszármazott újabb element-ek. A feldolgozást nehezíti, ezért nem szeretjük a kevert (mixed) element tartalmakat, amikor vegyesen van szöveg és leszármazott element a tartalomban.

<tagname attribute=”value”>CONTENT</tagname>

Ha nincs az element-nek tartalma (<tagname></tagname>), akkor azt rövidíthetjük úgy, hogy a nyitó tag végén jelezzük egy „/” (per) jellel, hogy rögtön az elem végére értünk, nincs tartalma (<tagname/>). Természetesen attribútumok lehetnek itt is.

### Nagyon jól formázott („Very well”-formed)

Azért, hogy a megoldásunk minél egyszerűbb legyen, vagyis ne kelljen figyelni a fölösleges space-kre, tabulátorokra, ne legyen probléma az előre olvasott karakterek elhagyásából, bevezetjük a nagyon jól formázott definíciót, ahol a jól formázottságon kívül még kell, hogy

1. az attribútumok között pontosan egy üres hely (space) legyen
2. ne legyen kevert (mixed) node, ahol egy elemen belül van szöveg másik elem és szöveg esetleg (<div>Egy <b>félkövér</b> szöveg</div> nem elfogadható.)
3. Az üres node lezáró „/” (per) jele rögtön az utolsó attribútum után következzen.
4. Záró tag-nél a „<” (kisebb) jel után rögtön a „/” (per) jel következzen.

## XML feldolgozása

### DOM

Dokumentum Objektum Model (Document Object Model), mely egy fa szerkezetben (struktúra) képzeli el a szöveget. A feldolgozáskor beolvassa a teljes fájlt a memóriába és egy fa adatszerkezetben tárolja. Az elemek elérése, programozása gyorsabb, hiszen közvetlenül a memóriában tudja bejárni a fát akár többször. De nagy a memória igénye.

### SAX

Egyszerű programozási interfész az XML-hez (Simple API for XML), mely az XML dokumentumot egy egyszerű szövegfájlnak tekinti, melyet szekvenciálisan dolgoz fel. Így ha egy elem megkeresése után egy korábbi elemre van szükség, akkor újra elkezdi elölről olvasni a fájlt. Lassabb vele a munka bizonyos esetekben (ha egyszer kell szekvenciálisan végigolvasni, nyilván nem lassabb.). Memóriaigénye kicsi (egy node adattartalmától függ). Szinte minden programnyelven van függvénykönyvtár, ami ezt megvalósítja. Ma mi is megvalósítjuk! Jellemzően az a megoldás, hogy van egy általános „parse”-olás, feldolgozás, ami callback hívásokkal a saját függvényeinket futtatja le. Több féle megoldás lehet: interface-n keresztüli elérés, absztrakt osztály, eseményvezérlés, … . Mi most a legegyszerűbbet, a függvények felülírását választjuk beágyazott, include fájl segítségével.

A következő eseményeket, alapműveleteket definiálja:

|  |  |  |
| --- | --- | --- |
| **Esemény/Művelet** | **Mikor** | **Paraméterek** |
| **startDocument** | A fájl/dokumentum elején hajtódik végre | nincs |
| **endDocument** | A fájl/dokumentum végén hajtódik végre | nincs |
| **startElement** | Egy nyitó tag olvasásakor fut le | elem neve, attribútum-lista, [névtér] |
| **endElement** | Egy záró tag olvasásakor fut le | elem neve, [névtér] |
| **characters** | Karakteres/szöveges adatok olvasásakor fut le. | Szöveg (olvasott karakterek) |

Ezeken kívül még párat, amikkel most nem foglalkozunk:

* processingInstruction - feldolgozási utasítás
* ignorableWhitespaces – mellőzhetőKarakterek
* skippedEntity – kihagyottEntitások

Sőt mi is írhatunk sajátot és felüldefiniálhatjuk például a kommentek kezelésére.

# SAX\_template

## Letöltés

<http://xml.inf.elte.hu/2009_10_2/szovegfeldolgozas_xml/SAX_template.zip>

<http://xml.inf.elte.hu/>

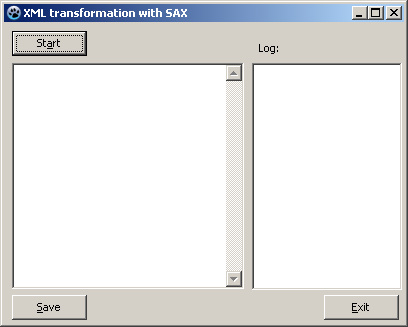
Oktatás / 2009/2010 Tavaszi félév / Szövegfeldolgozás - XML

SAX\_template.zip

## Fájlok

|  |  |
| --- | --- |
| **Fájl neve** | **Leírása** |
| unit1.pas | Főprogram a felületnek. **Nem kell módosítani!** |
| saxunit.pas | Az általános feldolgozó eljárást és a segédeljárásokat tartalmazza |
| SAX\_handler.inc | Az alapműveleti eseményekkor meghívott eljárásokat tartalmazza (fenti 5) |

## Felület



Napló bejegyzések helye

Eredmény helye

Eredmény mentése

Fájl helyének kiválasztása után

SAX feldolgozás indítása

Fájl kiválasztása után

Kilépés

## Típusok

Külön típust hoztunk létre az attribútum tárolására a név és érték külön tárolására.

TAttribute=Rekord

name:Szöveg

value: Szöveg

vége

Egy elemhez több attribútum tartozhat, így azokat tömbbe szervezzük.

TAttributeArray=Tömb(1..100:TAttribute)

Az attribútumok tömbjét kvázi listaként használjuk, vagyis az elemszámot még kezeljük.

TAttributeList= Rekord

count:egész;

item:TAttributeArray;

end;

## Változók

A fő változók a SAX osztályon belül a generált szöveg és a naplóbejegyzések tárolására használt szöveges változók, illetve a szöveges fájl.

parsedStr,log:Szöveg;

f\_xml:SzövegesFájl;

## Segédváltozók

Néhány segéd változóra is szükség lehet. Karakter előre olvasására vagy a szöveg karakterinek gyűjtésére.

ch:Karakter;

s1,s:Szöveg;

Az attribútum nevének és értékének tárolására külön, majd egyben az attribútumok listájában.

ename,value: Szöveg;

att\_list:TAttributeList;

Illetve az elválasztáskor az „=” (egyenlőség) jel pozíciójára.

p: egész;

Ciklusváltozóra.

i:egész;

A tag illetve node végének jelölésére.

endoftag,endofnode:Logikai;

## Eljárások és függvények

A fő változókhoz hozunk létre getter/setter függvényeket, melyeken keresztül lehet elérni a változót.

setString(fs: Szöveg);

setLog(fs: Szöveg);

getString: Szöveg;

getLog: Szöveg;

A fő változókhoz hozzá is fogunk fűzni adatot.

addString(fs: Szöveg);

addLog(fs: Szöveg);

Az XML fájl feldolgozását a következő általános eljárás végzi el.

parseXML(fname: Szöveg);

Szükségünk lesz parse-oláshoz a következő eljárásokra.

readForwardACharacter: Szöveg;

readToLess: Szöveg;

readToGreater: Szöveg;

readAWord(endoftag:Logikai;endofnode:Logikai): Szöveg;

readCDATASection: Szöveg;

readAttributesTo(att\_list:TAttributeList; endofnode:Logikai);

A következő eljárásokat fogjuk felülírni, mint saját események. Ezek a SAX\_handler.inc fájlban találhatóak.

startDocument;

endDocument;

startElement(ename: Szöveg;attr\_list:TAttributeList);

endElement(ename: Szöveg);

findCharacters(value: Szöveg);

A nyitó tag feldolgozásához segédfüggvény a következő.

getAttributeValue(attr\_list:TAttributeList;aname: Szöveg): Szöveg;

# Algoritmusok

## SAX feldolgozás megvalósítása (SAX\_handler.inc)

### startElement(ename:Szöveg;attr\_list:TAttributeList)

Az nyitó tag olvasásakor fut le.

elágazás

ename='example' esetén

//táblázat kezdete

addString(' <table border="1" cellpadding="0" cellspacing="0">');

addString(' <tr><td>ETR id</td><td>Name</td><td>E-mail</td></tr>');

ename='teacher' esetén

//sor eleje

line:=' <tr';

//ha van attributum, akkor kiolvassuk az „executive” értékét

ha attr\_list.count>0 akkor

value:=getAttributeValue(attr\_list,'executive');

//ha igazra („true”) van állítva, akkor a stílusosztály legyen „gray” és így majd szürke háttérrel jelenik meg.

ha value='true' akkor

line:=line+' class="gray"';

elágazás vége

elágazás vége;

line:=line+'>';

addString(line);

((ename='etrid') vagy (ename='name') vagy (ename='email')) esetén

addString(' <td>');

elágazás vége;

### endElement(ename: Szöveg)

Az záró tag olvasásakor fut le.

elágazás

ename='example' esetén

//táblázat vége

addString(' </table>');

ename='teacher' esetén

//sor vége

addString(' </tr>');

((ename='etrid') vagy (ename='name') vagy (ename='email')) esetén

//mező vége

addString(' </td>');

elágazás vége;

## getAttributeValue megvalósítása (saxunit.pas)

### getAttributeValue(attr\_list:TAttributeList;aname: Szöveg): Szöveg

Lineáris kereséssel megkeressük az „aname”-hoz tartozó rekordot a listában, majd ha megtaláljuk, akkor visszaadjuk a rekordbeli értéket.

value:='';

i:=1;

ciklus amíg ((i<=attr\_list.count) és (attr\_list.item[i].name<>aname))

inc(i);

ciklus vége;

ha (i<=attr\_list.count) akkor

value:=attr\_list.item[i].value;

elágazás vége;

getAttributeValue:=value;

## parseXML megvalósítása (saxunit.pas)

### parseXML(fname: Szöveg);

Ez az általános feldolgozó eljárás, mely a paraméterként megkapott nevű fájlból olvassa a karaktereket szekvenciálisan. A futás a fájl megnyitásával, a startDocument eljárás meghívásával kezdődik és a fájl lezárásával, az endDocument eljárás meghívásával fejeződik be. Először az első tag kezdetét ( az első „<” (kisebb) jelet) megkeressük, addig kiolvassuk a karaktereket. Itt kezdődik a feldolgozás. Addig folytatjuk, amíg nem érünk a fájl végére. Beolvassuk a tag első karakterét, ami alapján elágazunk és feldolgozzuk a nyitó tag-et, majd kiolvassuk a következő tag-ig („<” (kisebb) jelig) a karaktereket és azokat is feldolgozzuk. Visszatérve az elágazásra, ahol négy (4) lehetőség van.

1. A „/” (per) jel a záró tag-et jelenti, ahol ki kell olvasni a tag nevét és meg kell hívni az endElement eljárást.
2. A „?” (kérdőjel) jel feldolgozási utasítást jelent, amikor a nagyobb jelig beolvassuk a karaktereket, de tovább most nem dolgozzuk fel.
3. A „!” (felkiáltójel) jel speciális element-et jelent. Ilyenkor még a következő karaktert is beolvassuk és újból elágazunk ettől függően. Itt újra négy (4) lehetőség van.
   1. A „-„ (mínusz) jel megjegyzésre utal. Ekkor csak a nagyobb jelig beolvassuk a karaktereket, de tovább most nem dolgozzuk fel, mint a 2. esetben.
   2. A „D” betű a dokumentum típus definícióra utal. Ekkor ismét csak a nagyobb jelig olvassuk be a karaktereket, de tovább most nem dolgozzuk fel, mint az előbb.
   3. A „[„ (szögletes nyitó zárójel) jel a CDATA szekciót kezdi. Aminek lezárása a „]]>” (szögletes záró zárójel, szögletes záró zárójel, nagyobb jel) karaktersorozat. A szekció tartalmát beolvassuk és karaktersorozatként feldolgozzuk.
   4. Az összes többi eset ismeretlen a feldolgozás szempontjából. Így ismét csak a nagyobb jelig olvassuk be a karaktereket, de tovább most nem dolgozzuk fel, mint a b) esetben.
4. Más karakter esetén a nyitó tag nevének első kerekterét olvastuk, így folytatjuk a név beolvasását, majd, ha még nincs vége a tag-nek („endoftag”), akkor az attribútumokat is beolvassuk. Ha a nyitótag végén szerepel egy „/” (per) jel („endofnode”), akkor az olyan, mintha a záró tag-et is beolvastuk volna, mert üres element-ről van szó.

//Kiürítjük feldolgozott szöveget az új feldolgozás előtt.

setString('');

Nyit(f\_xml,fname);

//Dokumentum kezdete

startDocument;

//Kiolvassuk a fájl tartalmát az első ”<” (kisebb) jelig.

s:=readToLess;

ciklus amíg (nem fájl\_vége(f\_xml))

s:=readForwardACharacter;

elágazás

s='/' esetén

//Element vége

ename:=readAWord(endoftag,endofnode);

endElement(ename);

s='?' esetén

//Feldolgozási utasítás - kihagyjuk

s:=readToGreater;

s='!' esetén

//Speciális element

s:=readForwardACharacter;

elágazás

s='-' esetén

//Megjegyzés (<!--...-->)'

s:=readToGreater;

s='D' esetén

//DTD (<!DOCTYPE...>)'

s:=readToGreater;

s='[' esetén

//CDATA (<![CDATA[...]]>);

s:=readCDATASection;

ha s<>'' akkor

findCharacters(s);

elágazás vége

egyéb esetekben (különben)

//Ismeretlen speciális element

s:=readToGreater;

elágazás vége;

egyéb esetekben (különben)

//element kezdete, s tartalmazza a név első betűjét!

ename:=readAWord(endoftag,endofnode);

ename:=s+ename;

att\_list.count:=0;

ha nem endoftag akkor

readAttributesTo(att\_list,endofnode);

elágazás vége

//Element kezdete

startElement(ename,att\_list);

//Ha a nyitó tag végén van „/” (per) jel, akkor üres tag, és rögtön le kell zárni

ha endofnode akkor

//Element vége

endElement(ename);

elágazás vége

elágazás vége;

//Beolvassuk a karaktereket a következő (nyitó vagy záró) tag-ig, ami „<” (kisebb) jellel kezdődik, a trim függvény levágja az elejéről és végéről a fölösleges karaktereket

s:=trim(readToLess);

ha s<>'' akkor

findCharacters(s);

elágazás vége;

ciklus vége;

//Dokumentum vége

endDocument;

Zár(f\_xml);

## segéd eljárások megvalósítása (saxunit.pas)

### readForwardACharacter: Szöveg;

Egy karakter beolvasása (előre olvasás), majd szövegként visszaadjuk.

KaraktertOlvas(f\_xml,ch);

readForwardACharacter:=ch;

### readToLess: Szöveg;

Egy karakter előreolvasása majd a szöveghez fűzése addig, amíg a „<” (kisebb) jelet nem kapjuk. Az utolsó, már beolvasott karaktert elfelejtjük. Elhagyhatjuk, mert vagy szóválasztó vagy a tag végét jelző karakter.

s:='';

s1:=readForwardACharacter;

ciklus amíg ((nem fájl\_vége(f\_xml)) és (s1<>'<'))

s:=s+s1;

s1:=readForwardACharacter;

ciklus vége;

readToLess:=s;

### readToGreater: Szöveg;

Egy karakter előreolvasása majd a szöveghez fűzése addig, amíg a „>” (nagyobb) jelet nem kapjuk. Az utolsó, már beolvasott karaktert elfelejtjük. Elhagyhatjuk, mert vagy szó elválasztó vagy a tag végét jelző karakter.

s:='';

s1:=readForwardACharacter;

ciklus amíg ((nem fájl\_vége(f\_xml)) és (s1<>'>'))

s:=s+s1;

s1:=readForwardACharacter;

ciklus vége;

readToGreater:=s;

### readAWord(var endoftag:Logikai;var endofnode: Logikai): Szöveg;

Egy karakter előreolvasása majd a szöveghez fűzése addig, amíg a szóvégéhez nem érünk, vagyis a „>” (nagyobb), a „/” (per) vagy „ „ (space) jelet nem kapjuk. Az utolsó, már beolvasott karaktert még feldolgozzuk, hogy vége van-e tag-nek vagy node-nak, majd elfelejtjük. Elhagyhatjuk, mert vagy szó elválasztó vagy a tag végét jelző karakter. Az endofnode jelzi, hogy nem lesz záró tag vagyis a „startElement” után az „endElement”-et is rögtön meg kell hívni.

endofnode:=HAMIS;

s:='';

s1:=readForwardACharacter;

ciklus amíg ((nem fájl\_vége (f\_xml)) és (s1<>'>') és (s1<>' ') és (s1<>'/'))

s:=s+s1;

s1:=readForwardACharacter;

ciklus vége;

ha s1='/' akkor

endofnode:=true;

s1:=readToGreater;

endoftag:=true;

különben

endoftag:=(s1='>');

elágazás vége;

readAWord:=s;

### readCDATASection: Szöveg;

Hasonlít a szó beolvasására (readAWord), de 3 karaktert kell előre olvasni, és „]]>”-nél van vége.

Házi feladat!

### readAttributesTo(var att\_list:TAttributeList;var endofnode: Logikai);

Hátul-tesztelős ciklusban, hogy egyszer mindenképpen lefusson, beolvasunk egy szót (név=”érték” formátumban), majd azt szétvágjuk a névre illetve értékre, amit az attribútumok listájába töltünk. Továbbadjuk a szóolvasból kapott „endofnode” paraméter értékét.

ciklus

//Olvasunk egy szót.

s:=readAWorld(endoftag,endofnode);

//Növeljük az attribútumok számát

att\_list.count:=att\_list.count+1;

//Az „=” (egyenlőség) jel pozícióját megkeressük

p:=Pos('=',s);

//A szövegből kimásoljuk az attribútum nevét tartalmazó részt. Az „=” (egyenlőség) jelet már nem.

att\_list.item[att\_list.count].name:=Szövegrészt\_másol(s,1,p-1);

// A szövegből kimásoljuk az attribútum értékét tartalmazó részt. Az „”” (macskaköröm) jeleket nem.

att\_list.item[att\_list.count].value:=Szövegrészt\_másol(s,p+2,length(s)-p-2);

amíg nem (fájl\_vége(f\_xml) vagy endoftag);

# Jegyzeteim

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………

……………………………………………………………………………………………………………………………………………………………………