XML és XSLT
(a színfalak mögül)

Írta: Nagy Tamás

Motiváció

Ez a dokumentum eredetileg egy előadás írott változatának készült. Már az előadásra való felkészülés során, több könyv és Internetes oldal elolvasása és átböngészése során nyilvánvalóvá vált a számomra, hogy a legtöbb információforrás csak az XML legalapvetőbb képességeivel, lehetőségeivel foglalkozik. Meglehetősen sokat kellet keresgélni, amíg lassan összeállt a kép, és kezdtem az XML mélyebb rétegeit is megérteni. Pedig - így utólag - az egész nem bonyolult, sőt kijelenthetem, kifejezetten egyszerű. Csak éppen nincs egy tisztességes, összeszedett, minden részletre kiterjedő leírás hozzá. Azt nem állíthatom, hogy ez az lett. Bár törekedtem arra, hogy minden részletet megragadjak, és főleg, hogy olyan kérdésekre adjak választ ebben a segédletben, amelyeket egyébként nehezen lehet elő bányászni.

Tartalomjegyzék

1. Bevezetés

Az XML azt jelenti "eXtensible Markup Language" magyarul bővíthető jelölő nyelv. Ez egy leíró nyelv, melynek célja elsősorban adatok, információk, és ezek struktúráinak leírása. Adatok információk leírására szolgáló leíró nyelv már eddig is létezett: az SGML (Standard Generalized Markup Language - Szabványos Általánosított Jelölő Nyelv).

Ez egy rendkívül komplex és teljesen általánosított jelölőnyelv. Elterjedését is pont ez gátolta meg, rendkívüli komplexitása miatt ugyanis nagyon nehéz implementálni SGML értelmezőket illetve alkalmazásokat. Az SGML tapasztalataira és irányelveire építve hozták létre a HTML-t, amely egy korlátozott céllal és funkcionalitással bíró jelölőnyelv, egy sokkal egyszerűbb felépítéssel. Ez az egyszerűség tette lehetővé a HTML világméretű elterjedését.

Az XML-t az SGML és a HTML tapasztalataira alapozva hozta létre a webes szabványok kidolgozásáért felelős szervezet a W3C (World Wide Web Consortium). Az XML a HTML-éhez hasonlóan egyszerű felépítésű, azonban annál általánosabb funkciójú, rugalmasabb, mégis szintaktikai szempontból szigorúbb is, ami leegyszerűsíti az ipari méretű fejlesztést és hibakezelést.

Az XML kidolgozásakor elsődleges volt az egyszerűség, a széleskörű alkalmazhatóság beleértve az Interneten való felhasználhatóságot is, az SGML kompatibilitás, valamint, hogy a nyelvnek formálisnak és konzekvensnek kell lennie. Nem volt szempont a fejlesztés során a tömör adatábrázolás. (megjegyzés: A tömörséget ugyanis megfelelő tömörítők használatával biztosítani lehet.)

2. Felépítés

Az XML-ben az adatokat elemek (TAG-ek) jelölik. Az elemeket az XML (A HTML-hez hasonlóan) kisebb nagyobb jelekkel ("<", ">"), mint határoló karakterekkel jelöli meg, és a határoló elemek közötti szó - címke - írja le magát az elemet. Példa:

```
<Első_elem>
```

A HTML-ben az egyes felhasználható elemek köre zárt, az elemek és jelentésük előre meghatározott. Az XML-ben ezzel szemben nincsenek meghatározva sem az egyes elemek nevei, sem jelentésük, sem alkalmazásuk módja. Ez lehetővé teszi, hogy mindenki a saját maga számára legmegfelelőbb, leginkább beszédes, vagy éppen a saját meglévő rendszereihez leginkább illeszkedő elnevezéseket válasszon.

Példa:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<megjegyzés>
<címzett>Tamás</címzett>
<feladó>Jani</feladó>
<fejléc>Emlékeztető</fejléc>
<törzs>Ne felejtsd el elzárni a vizet! </törzs>
</megjegyzés>
```

Minden XML dokumentumnak egy vezérlési utasítással kell kezdődnie, amelyben legalább azt meg kell adni, hogy a dokumentum melyik XML verzió szerint készült.

Vezérlési utasítás:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
```

Ebben a dokumentumban a felhasznált kódlap is meg van adva.

Az XML dokumentumban található adatok egy fa struktúrát alkotnak. Minden dokumentum egy gyökér elemből indul, a fenti példában ez a <megjegyzés> elem.

```
A megjegyzés dokumentum fa nézetben:
megjegyzés
      címzett              (értéke: Tamás)
      feladó               (értéke: Jani)
      fejléc               (értéke: Emlékeztető)
      törzs                (értéke: Ne felejtsd...)
```

Egy XML dokumentumnak csak egy gyökere lehet, de minden dokumentumnak **kötelezően** rendelkeznie kell egy gyökér elemmel.

Minden elemhez kötelezően tartozik egy bezáró pár, ami az elem nyitó párjával együtt egyértelműen azonosítja (közrefogja) az elemhez tartozó tartalmat (adatokat). Egy elem bezáró párja a nyitó pártól abban különbözik, hogy a záró elemben a nevét jelző címke előtt egy per ("/") jel szerepel.

```
Elem záró párral: <megjegyzés> adatok... </megjegyzés>
```

Ha egy elem esetében nincs értelme a bezáró párnak, azt az elem vége előtti "/" jellel kell jelezni, például: <BR/>. Egy XML dokumentumban a nyitó és záró elemre a hagyományos zárójelezés szabályai érvényesek, a HTML-ben megengedett ... szerkezet nem érvényes. Ez néhány esetben a HTML-énél bonyolultabb szerkezetet eredményez, ugyanakkor ez a felépítés mind az értelmezők, mind világos dokumentum szerkezetek létrehozásakor előnyösebb.

A HTML-ben megszokott forma (XML-ben nem használható):

```
•   Ez a szöveg <b> félkövér <i> ez a rész dőlt is </b> ez már csak dőlt </i>
```

Ugyanez XML-ben:

```
•   Ez a szöveg <b> félkövér <i> ez a rész dőlt is </i></b><i>ez már csak dőlt </i>
```

Az XML dokumentum bővíthetősége azt jelenti, hogy új elemet és ezáltal új adatot adhatunk az adathalmazhoz anélkül, hogy a régi alkalmazás működése ettől megváltozna (elromlana). Természetesen új elemek hozzáadásakor azok kezelésére az alkalmazást fel kell készíteni, de önmagában az új elem hozzáadása az addigi munkában fennakadást nem okoz.

Ami minden XML dokumentum esetén kötelező:

- Vezérlési utasítás + verzió információ,
- Gyökér elem,
- Kötelező bezáró pár,
- Zárójelezési szabály.

3. Karakterek, elnevezések

Az XML esetén az egyes elemek neveiben illetve az adatokban tetszőleges karakterek felhasználhatók, azzal a kitétellel, hogy az elnevezéseknek és a karakterek felhasználásának meg kell felelniük bizonyos szabályoknak.

- A szabvány szerint az XML megkülönbözteti a kis és nagy betűket (case sensitive). Léteznek megvalósítások, amelyek ebben a tekintetben nem követik a szabványt, de a legtöbb (és a szabványos) implementációk ilyenek.
- A HTML-lel ellentétben az XML értelmezők nem veszik ki az adatokból a többszörös kihagyás jellegű (white space) karaktereket.
- Elemek nevei tartalmazhatnak bármilyen karaktert, a következő szabályok figyelembevételével:
  - Elem név nem tartalmazhat kihagyást (space).
  - Elem név nem kezdődhet számmal valamint aláhúzás karakterrel.
  - Elem név nem kezdődhet az XML karakter sorozattal
  - Elem névben nem ajánlott a mínusz (-) és a pont (.) karakter valamint a szintén nem ajánlott a kettőspont (:) mert speciális jelentése van.

Nincs fenntartott szó, a fentieknek megfelelő bármilyen karakter sorozat szerepelhet egy elem nevében.

Adatok neveiben szintén tetszőleges karakterek lehetségesek azzal a megkötéssel, hogy az elemzett adatokban (parsed data) a kisebb a nagyobb valamint az idéző jeleket kerülni kell – ezek karakter kódokkal vagy speciális egyedek (< > & stb.) felhasználásával adhatók meg.

4. Névterek (Namespaces)

A névterek használata lehetővé teszi a név konfliktusok elkerülését két azonos elnevezésű és különböző célú elem egyazon XML dokumentumban való használata esetén. Név konfliktusok számos okból fordulhatnak elő, elsősorban azért, mert az XML nem tesz megkötést az elemek neveire vonatkozóan, ezért könnyen előfordulhat, hogy amikor két különböző forrásból származó XML dokumentumból hoz létre valaki egy új XML dokumentumot, akkor az új dokumentumban név konfliktus (ok) adódnak. Különösen fontos probléma ez azért, mert egyre inkább tendencia, hogy a különböző újonnan létrehozott Internetes technológiákat az XML valamilyen speciális felhasználásai (pl.: XSLT, XML Schema vagy XSD), és ha valaki e technológiák valamelyikét a saját alkalmazásában felhasználja, valamint a technológiával azonos elemneveket használ akkor névkonfliktus lép fel.

4.1 Név konfliktus.

```
<Dokumentum>
 <méret>4 adat</méret>
 <típus>XML</típus>
 <doc>
   <ruha>
    <típus>póló</típus>
    <szín>sárga</szín>
    <méret>XXL</méret>
    <forma>hagyományos</forma>
   </ruha>
 </doc>
</Dokumentum>
```

Ebben a példában egy dokumentum keret van, amiben elvileg tetszőleges dokumentum lehet beágyazva. Itt egy XML típusú, ruháról szóló 4 adatot tartalmazó dokumentumot tartalmaz, amely azonban szintén rendelkezik <típus> és <méret> elemekkel, ez pedig név konfliktust okoz.

4.2 Megoldás név előtéttel (prefix).

```
<doc:Dokumentum>
          <doc:méret>4 adat</doc:méret>
          <doc:típus>XML</doc:típus>
          <doc:doc>
          <adat:ruha>
                    <adat:típus>póló</adat:típus>
                    <adat:szín>sárga</adat:szín>
                    <adat:méret>XXL</adat:méret>
                    <adat:forma>hagyományos</adat:forma>
          </adat:ruha>
          </doc:doc>
</doc:Dokumentum>
```

A dokumentum különböző részeit különböző előtéttel egészítettük ki, így biztosítottuk az egyediséget.

4.3 Megoldás névtér használatával.

Névtér használata annyiban különbözik az előtét esetén megismerttől, hogy az előtét első előfordulásakor definiálni kell, hogy az egy névtér, és hozzá kell rendelni egy egyedi névtér nevet.

```
<doc:Dokumentum xmlns:doc="http://www.aut.bme.hu/docns.html">
      <doc:méret>4 adat</doc:méret>
      <doc:típus>XML</doc:típus>
      <doc:doc>
      <adat:ruha xmlns:adat="http://www.aut.bme.hu/ruhans.html">
                <adat:típus>póló</adat:típus>
                <adat:szín>sárga</adat:szín>
                <adat:méret>XXL</adat:méret>
                <adat:forma>hagyományos</adat:forma>
      </ adat:ruha>
      </doc:doc>
</doc:Dokumentum>
```

Ebben a példában definiáltunk két névteret. A két névtér nevének egy-egy URL-t adtunk meg. Az XML értelmező ezt az URL-t nem használja semmilyen célra, mindössze egy egyedi névként tekinti (tehát ez lehetne egy tetszőleges szó is), bár sok esetben az XML dokumentumok készítői az URL által hivatkozott oldalon helyeznek el információt az adott névteret illetően, illetőleg előfordul, hogy valamely technológia (pl.: XSLT, XHTML, vagy HTML) névterének nevét a technológia értelmezője (például egy böngésző) felhasználja a

technológia azonosítására (hogy képes-e, illetőleg hogyan képes értelmezni az adott technológiát). Emiatt van, hogy az Internet Explorer egy adott XSLT-t a megadott névtér névtől függően eltérően képes kezelni - vagyis, ha elírjuk a névtér nevét, akkor előfordulhat, hogy emiatt nem működik a stíluslap.

5. Tulajdonságok

Akárcsak a HTML-ben az XML-ben is rendelkezhetnek az egyes elemek tulajdonságokkal, azonban az XML-ben a attribútumok értékeinek mindig kötelezően idézőjelek között kell lenniük.

```
Példa:
<megjegyzés dátum="99/12/24">
```

Az idézőjel lehet egyszeres és kettős idézőjel is.

A tulajdonságok hasonló szerepet töltenek be, mint az adott elem gyermek elemei: valamilyen információval vagy adattal bővítik annak jelentéstartalmát.

Példa ugyanannak az információnak két féle kezelésére:

```
1)

<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>


2)

<person>
  <sex>female</sex>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

Az iménti két példából mindkettő szabványos, mindkettő ugyanazt jelentheti. Nincs előírás, hogy mikor kell egy adott információ vagy adat tárolásához gyermek elemet és mikor tulajdonságot használni, azonban a kettő kezelése közötti különbség rá világít a gyermek elemek és a tulajdonságok használatának irányelveire.

- A tulajdonságok nem tartalmazhatnak egyszerre több adatot (a gyermek elemek igen).
- A tulajdonságok nem bővíthetők olyan egyszerűen, mint a gyermek elemek.
- A tulajdonságok adatai nem rendezhetők struktúrákba (a gyermek elemek adatai igen)
- A tulajdonságokat nehezebb kezelni a programokban.
- A tulajdonságok helyességet nehezebb ellenőrizni

Mindezek alapján tulajdonságokat akkor érdemes használni, ha azok nem az adatok, hanem az azokat feldolgozó alkalmazások szempontjából jelentenek plusz információt. Nincs előírás, hogy mikor kell egy információt adatként vagy attribútumként megadni, de ami adatként felhasználandó, azt érdemes adatként megadni.

Példa tulajdonság használatára:

```
<image type="gif" name="picture.gif"/>
```

Ebben a példában a felhasználó számára nem lényeges adat sem a kép neve, sem a típusa, - csak maga a kép - azonban a kép megjelenítéséhez a megjelenítést végző alkalmazásnak szüksége van ezekre az információkra.

Egy másik jellemző alkalmazási példa, amikor tulajdonságokat érdemes használni, ha egy adat rekord halmazban az egyes rekordokat egyedi azonosítóval kell ellátni. Ekkor az azonosító a felhasználó felé ismét nem lényeges adat, azt kizárólag az adatokat feldolgozó alkalmazás használja.

6. Dokumentum séma definíció

Egy XML dokumentum tetszőleges elemeket és tulajdonságokat tartalmazhat, valamint tetszőleges felépítéssel rendelkezhet. Mindezek a paraméterek meghatározzák a dokumentum típusát, amelyet leírva a dokumentum típus definícióját, vagy sémáját kapjuk.

Egy dokumentum típus definíciója számos módon felhasználható. Felhasználható több, különböző forrásból származó XML dokumentum „kompatibilitásának” a vizsgálatára (hogy vajon a dokumentumok azonos

típusúak, így ugyanabban az alkalmazásban egyaránt felhasználhatóak-e). Felhasználható annak meghatározására, hogy egy készítendő dokumentumnak milyen típussal kell rendelkeznie, milyen konvenciókat kell betartania, hogy egy adott alkalmazásban felhasználható legyen, valamint a típus definíció alapján vizsgálható egy dokumentum érvényessége (nem gépelt-e el valamit a készítője).

Jelenleg három jelentős séma leíró módszer létezik. A jelenleg legelterjedtebb a DTD (Document Type Definition). Legfontosabb előnye, hogy szabványos, készen áll, és eddig mindenki, aki típus leírást használt az XML dokumentumaihoz ezt használta. Hátránya, hogy nem XML alapú nehéz megtanulni, és bonyolult, ráadásul mindezek miatt a jövőben valószínűleg el fog tűnni.

Az első elkészült XML alapú séma leírási módszer az XDR. (XML alapú annyit tesz - maga a leírás is egy XML dokumentum, a nyelvi elemek XML elemek is egyben.). A Microsoft fejlesztette ki, mind közül a legkönnyebben megtanulható, végleges, a Microsoft eszközei széles körűen támogatják, ugyanakkor nem szabványos, más vállalatok termékei csak elvétve ismerik.

Az XSD (XML Schema Definition) vagy XML Schema szabványos, széles körűen támogatott, általános séma leíró nyelv, azonban legfőbb hátránya, hogy még nincs kész. Vélhetően hamarosan ez lesz az XML-hez hivatalosan ajánlott dokumentum típus leíró nyelv.

6.1 DTD (Document Type Definition)

A DTD egy fajta szabványos típus-definíció. Az SGML nyelvhez készült, de az XML esetén is kiválóan használható, ma az egyik legelterjedtebb típus-definíciós módszer.

Példa:

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

6.1.1 Elemek

A dokumentum típusának definíció megadásakor a definíció nevének a gyökér elem nevével kell megegyeznie. Ezután következik az egyes elemek megadása. Először a gyökér elemet kell megadni, majd a gyermek elemek definíciójára kerül sor.

A példában a "note" elem definíciójakor meghatározásra kerül, hogy négy gyermek elemmel rendelkezik, ez hordozza a struktúra információit. Ugyanitt adható meg az is, ha valamely gyermek elemből több is előfordulhat a dokumentumban.

```
<!ELEMENT note (to+,from,heading,body)>
```

A pluszjel (+) a "to" elemnév után azt jelezheti, hogy több ilyen elem (itt címzett) is lehetséges a dokumentumban.

Az egyes elemek tartalmazhatnak további gyermek elemeket, elemzett (#PCDATA – Parsed Character Data), vagy nem elemzett (#CDATA - Character Data) adatokat. A nem elemzett adatok adatrészét az XML elemző nem vizsgálja, nem értelmezi, ezért ott tetszőleges karakterek – például a kisebb vagy a nagyobb jel - korlátozás nélkül használható.

6.1.2 Tulajdonságok

A DTD a tulajdonságok leírását is lehetővé teszi a következő módon.

```
<!ATTLIST elem-név tulajdonság-név tulajdonság-típus alapértelmezett-érték>
```

Példa:

```
<!ATTLIST megjegyzés dátum #CDATA "99/12/24">
```

Az elem-név annak az elemnek a neve, amelyhez a tulajdonság tartozik. A tulajdonság-név és az alapértelmezett-érték egyértelmű, a tulajdonság típus pedig a következők valamelyike lehet:

Típus	**Magyarázat**
CDATA	Nem ellenőrzött karakter adat
(en1|en2|..)	Az érték egy számozott lista valamely eleme lehet
ID	Az érték egy egyedi azonosító
IDREF	Az érték egy másik elem egyedi azonosítója
IDREFS	Más elemek egyedi azonosítóinak listája
NMTOKEN	Az érték egy érvényes XML név
NMTOKENS	Az érték érvényes XML nevek listája
ENTITY	Az érték egy egyed
ENTITIES	Az érték egyedek listája
NOTATION	Az érték egy megjegyzés neve
xml:	Az érték előre definiált xml: érték

6.1.3 Egyedek

A DTD-ben lehetőség van egyedek megadására. Az egyedek kezelése nagyon hasonlít a C nyelv **#include** illetve **#define** lehetőségeire, lehetővé teszi külső erőforrásban levő, vagy előre definiált adatok (szövegek, XML dokumentum részletek, stb.) újra felhasználását egy dokumentumban. Az egyedeket az **&egyed-név;** formában lehet megadni a dokumentumban, ahol az egyed helyett a hozzá rendelt karakter vagy szöveg meg.

Az egyedeket három csoportra oszthatjuk.

6.1.3.1 Előre definiált egyedek.

Egyed	Egyed név
Kisebb jel (<)	Lt
Nagyobb jel (>)	Gt
Angol és – AND – jel (&)	Amp
Egyszeres idézőjel (')	Apos
Dupla idézőjel (")	Quot

6.1.3.2 Normál egyedek

Megadása:

```
<!ENTITY egyednév "hozzá rendelt szöveg">
```

példa:

```
<!ENTITY IBM "International Business Machines">
```

Ekkor, ahol a dokumentumban az **&IBM;** karaktersorozat található, ott az International Business Machines szöveg jelenik meg.

6.1.3.3 Paraméter egyedek

```
<!ENTITY % paraméter-egyed-név "hozzá rendelt szöveg">
```

A paraméter egyedek abban különböznek a normál egyedektől, hogy csak a DTD-ben használhatók (a DTD felépítésének megkönnyítésére), az XML dokumentumban nem. Hivatkozni rá a **%paraméter-egyed-név;** utasítással lehet.

A külső egyedek a belső egyedektől annyiban különböznek, hogy a deklaráció tartalmazza a SYSTEM kifejezést, és a hozzá rendelt szöveg helyett egy erőforrás azonosító (URI/URL) mutat a bemásolandó adatra.

```
<!ENTITY ruha SYSTEM "http://www.aut.bme.hu/ruha.xml">
```

6.1.4 A DTD megadása

A dokumentum leírót az XML dokumentumban beágyazva, illetve egy külső fájlban is meg lehet adni.

Példa DTD XML-en belüli megadására:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE megjegyzés [
<!ELEMENT megjegyzés (címzett+, feladó, fejléc, törzs)>
<!ATTLIST megjegyzés dátum CDATA #REQUIRED>
<!ELEMENT címzett (#PCDATA)>
<!ELEMENT fejléc (#PCDATA)>
<!ELEMENT feladó (#PCDATA)>
<!ELEMENT törzs (#PCDATA)>
]>
<megjegyzés dátum="01/05/30">
      <címzett>Tamás</címzett>
      <címzett>József</címzett>
      <feladó>Jani</feladó>
      <fejléc>Emlékeztetï</fejléc>
      <törzs>Ne felejtsd el elzárni a vizet! </törzs>
</megjegyzés>
```

Lehetőség van a DTD külső (az XML fájlon kívüli) megadására is, ekkor a DOCTYPE elemben jelezni kell, hogy a leírás egy külső fájlban található:

```
<!DOCTYPE note SYSTEM "note.dtd">
```

Ahol a note.dtd fájl tartalmazza a definíciót:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

7. Megjelenítés

Az XML adatok, információk vagy struktúrák leírására szolgáló leíró nyelv. Önmagában nem foglalkozik az adatok megjelenítésének problémájával. Erre számos más módszer létezik. Kezelhető egy XML dokumentum HTML forrásban, különböző stíluslapok segítségével, illetve tetszőleges, Internettől, böngészőtől független alkalmazás fejleszthető a dokumentumok kezelésére és megjelenítésére.

7.1 HTML

Egy HTML dokumentumban XML állományt az <XML> Tag segítségével lehet felhasználni.

Példa XML forrás:

```
<?xml version="1.0"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
</CATALOG>
```

Példa HTML állomány:

```
<html><body>
<xml src="cd_catalog.xml" id="cdcat">
</xml>

<table datasrc="#cdcat" width="100%" border="1">
<thead>
<th>Title</th><th>Artist</th><th>Year</th>
</thead>
<tr align="left">
<td><span datafld="TITLE"></span></td>
<td><span datafld="ARTIST"></span></td>
<td><span datafld="YEAR"></span></td>
</tr></table>
</body></html>
```

Az XML elem fontos paramétere az SRC, amely a forrás XML állományt adja meg. Szintén elhagyhatatlan az ID paraméter, ez alapján lehet ugyanis az állományban az XML adatokra hivatkozni. Ezután a táblázatban megadható az adatforrás azonosítója (datasrc paraméter), majd az egyes adatokra az XML elemnév segítségével lehet hivatkozni.

7.2 CSS (Cascading Style Sheets)

Stílus lapok segítségével is meg lehet jeleníteni XML állományok tartalmát. Ekkor a stíluslapban az egyes XML elemekhez kell stílusokat rendelni. Az előző pont példa XML állományát figyelembe véve egy lehetséges stíluslap lehet a következő:

```
CD
{
color: green;
border: '1pt solid black'
page-break-after:always;
}
TITLE
{
color: red;
background-color:#EFE7D6;
border: '1pt solid black'
}
ARTIST
{
color: #FFFFFF;
background-color:#000000;
}
```

Természetesen ahhoz, hogy az XML állomány a megjelenéshez használjon egy stíluslapot, egy vezérlési utasítást kell beszúrni az adatok elé:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

7.3 XSL (Extensible Stylesheet Language)

A stíluslapok használata számos esetben elegendő lehet a megfelelő megjelenés létrehozásához, de nem ad lehetőséget olyan dokumentumok megjelenésének megvalósítására, ahol az adatok nem a megjelenítésnek megfelelően találhatók meg, például mert bizonyos adatok csak közvetetten határozhatók meg, vagy, mert az adatok struktúrája nem megfelelő a kívánt megjelenéshez. További problémát jelent a stíluslapokkal kapcsolatban, hogy a CSS egy a HTML-hez kifejlesztett technológia.

A megjelenés rugalmas megvalósítása érdekében kidolgoztak egy kifejezetten az XML igényeihez igazodó stílus leíró nyelvet, amely képes egy XML állomány tetszőleges megjelenítésére, ráadásul a megjelenést is XML formában egy XML dokumentum segítségével írja le, ez az XSL.

Az XSL lehetőségeivel külön fejezet foglalkozik.

8. XPath

Az XSLT és az XPointer szabványokhoz szükség volt egy módszerre, amelynek a segítségével leírható egy XML dokumentum adott része, ezért jött létre az XPath nyelv. Szintaxisa nem XML alapú, az XML dokumentumban való navigáción kívül lehetőséget ad logikai, számtani és string műveletek elvégzésére is. Egy XML dokumentum adott részének elérésén túl az XPath nyelv alkalmas annak leírására is, hogy egy csomópont megfelel-e egy adott mintának.

Az XPath értelmezésében egy XML dokumentum 7 féle részből állhat, ezek: Gyökér csomópontok (root nodes), csomópontok (element nodes), szöveges csomópontok (text nodes), tulajdonságok (attribute nodes), névterek (namespace nodes), vezérlési utasítások (processing instruction nodes) és megjegyzések (comment nodes).

Egy XPath kifejezésnek az eredménye lehet csomópontok egy csoportja, logikai érték (igen vagy nem), szám vagy szöveg.

Egy kifejezés kiértékelésekor rendszerint figyelembe vevődik az aktuális helyzet, amit az XSLT vagy XPointer értelmező határoz meg. Az aktuális helyzetet több információ együttesen írja le, például az aktuális csomópont, annak pozíciója, mérete, stb.

8.1 Elérési-út (location path)

Minden elérési-út lépésekből áll (location steps), mely lépések egymástól egy „/” jellel vannak elválasztva, ezen lépések összessége határozza meg az XML dokumentum adott pontjaihoz vezető utat. Ez nagyon hasonlít a fájl rendszerben megszokott elérési-utakhoz, ahol az egyes könyvtárak jelentik a lépéseket, itt azonban az egyes lépések jóval komplexebbek is lehetnek az XML dokumentum elemeinél, hiszen akár több elemet is jelölhetnek.

Akárcsak a fájl rendszerek esetén, itt is két féle útvonal leírás létezik. Az egyik a relatív, ahol az elérési út kiinduló pontja az aktuális pozíció, a másik az abszolút leírás, ahol a leírás egy „/” jellel kezdődik és a kiindulási pont az XML dokumentum gyökere (root). Az abszolút leírást lehet úgy tekinteni, mint egy olyan relatív leírást, ahol az aktuális pozíció ideiglenesen – erre az egy számításra – a gyökérre állítódik át.

Egy elérési-út által elérhető csomópontok körének meghatározásakor az elérési út értelmezése mindig balról jobbra történik. Az első lépés során a lépés kiinduló pontja az aktuális – vagy abszolút leírás esetén a gyökér – elem, és a lépés során kiszámítódik mindazon csomópontok köre, amelyet az adott lépés leír. A „child::Piros” kifejezés tehát az összes olyan elemet jelenti, amelynek a neve „Piros” és amelynek a szülője az aktuális elem. Ha a teljes elérési-út több lépésből áll, akkor a következő lépés kiinduló pontja - a lépés esetén értelmezett aktuális elem – az összes az előző lépésben elért elemek halmaza lesz, és az ő általa elért elemek köre ehhez képest relatív. A „child::Piros/child::Kék” elérési út tehát az összes olyan „Kék” elemet jelenti, amelynek szülő eleme „Piros”, és amelynek nagyszülője az aktuális elem (vagyis az aktuális elem összes piros gyermekének az összes kék gyermeke). Ha csak egy ilyen van, akkor ez az elérési út egy konkrét elemet jelöl, ha több is van, akkor több elemet is leír.

8.2 Lépések (location steps)

Egy XPath lépés három részből áll, felépítése a következő:

irány :: csomópont típus [feltételes kifejezés]

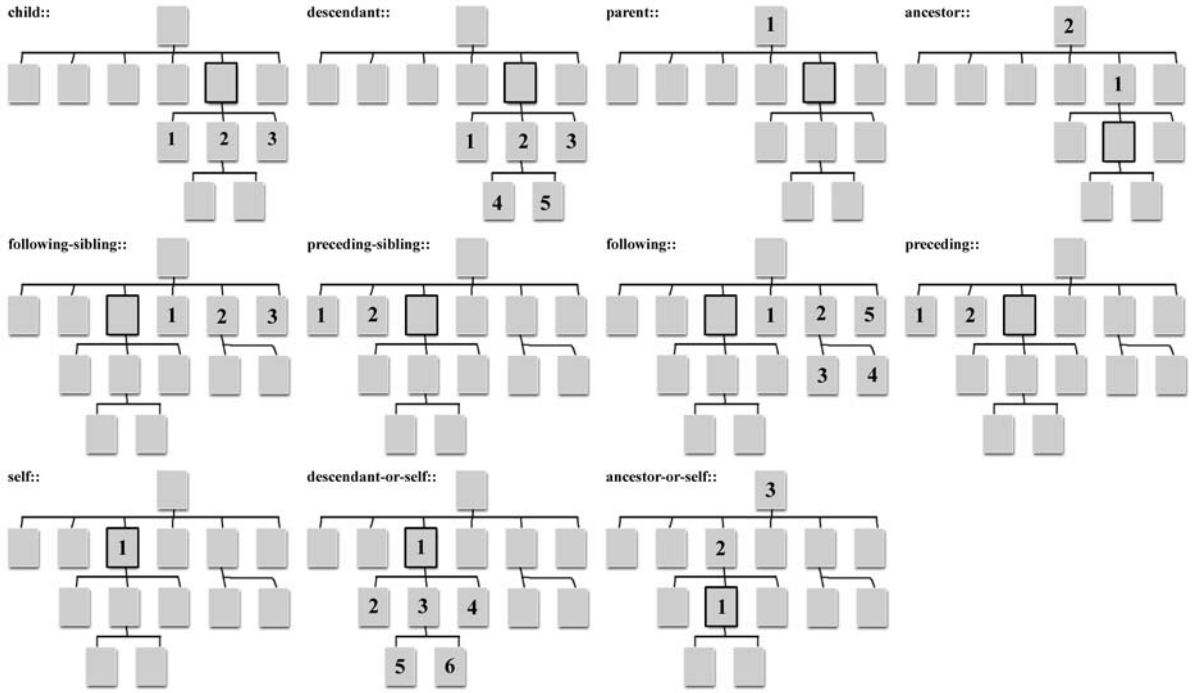
8.2.1 Irányok

Az irány, a lépéssel megjelölni kívánt csomópontok és az aktuális elem viszonyát írja le.

Lehetséges irányok

child::	**Gyermek.** Az aktuális elem gyermek eleme. Amennyiben nincs irány megadva, akkor ez az alapértelmezés szerinti irány.
descendant::	**Leszármazott.** Az aktuális elem összes leszármazottja (a gyermekei és azoknak is a gyermekei rekurzívan).
parent::	**Szülő.** Az aktuális elem szülő elemét jelöli ki.
ancestor::	**Felmenő.** Az aktuális elem összes felmenőit jelöli ki. (A saját szülőjét és annak a szülőjét is rekurzívan, így természetesen szükségszerűen mindig a gyökér elemet is.)
following-sibling::	**Következő testvér.** Az aktuális elem következő testvérét jelöli ki. Ha az aktuális csomópont tulajdonság vagy névtér, a következő elem halmaz üres.
preceding-sibling::	**Előző testvér.** Az aktuális elem előző testvérét jelöli ki. Ha az aktuális csomópont tulajdonság vagy névtér, az előző elem halmaz üres.
following::	**Következők.** Az aktuális elemet követő minden elem a dokumentum sorrendjében.
preceding::	**Megelőzők.** Az aktuális elemet megelőző minden elem a dokumentum sorrendjében.
attribute::	**Tulajdonság.** Az aktuális elem tulajdonságait jelöli ki. Üres, ha az aktuális elem nem egy csomópont.
namespace::	**Névtér.** Az aktuális elem névtereit jelöli ki. Üres halmaz, ha az aktuális elem nem egy csomópont
self::	**Önmaga.** Ez az aktuális elemet jelöli ki.
descendant-or-self::	**Leszármazottak vagy önmaga.** Az aktuális elemet és a leszármazottait jelöli.
ancestor-or-self::	**Felmenő vagy önmaga.** Az aktuális elemet és a felmenőit jelöli.

Az XPath lépések irányai (Mindig az aktuális elem van bekeretezve):



8.2.2 Típusok

Mint már szerepelt, az XPath 7 féle csomópont típust különböztet meg. Ebből három alap típus – névtér, tulajdonság, csomópont –, és négy a csomópont alaptípus speciális esete – szöveges csomópont, megjegyzés, gyökér, vezérlési utasítás. Az XPath-ban minden iránynak van egy elsődleges típusa, ez tulajdonság, ha az aktuális irány tulajdonság, névtér, ha az aktuális irány névtér, és minden más esetben csomópont.

Egy típust két módon lehet megadni, vagy név megadásával, vagy csomópont típus kiválasztó utasítások segítségével.

Lehetséges csomópont típusok

„név”	A „név” által megadott nevű tulajdonságot (attribute:: irány esetén), vagy elemeket választja ki.
„*”	Az adott iránynak megfelelő bármilyen nevű, tehát az adott iránynak megfelelő összes elemet kiválasztja.
comment()	Az adott iránynak megfelelő minden megjegyzés elemet kiválasztja.
text()	Az adott iránynak megfelelő minden szöveges elemet kiválasztja.
processing-instruction()	Az adott iránynak megfelelő minden vezérlési utasítást kiválasztja.
node()	Az adott iránynak megfelelő minden elemet a típusától függetlenül, kiválasztja.

Legyen adott a következő XML dokumentum.

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="pelda 01.xsl"?>
<Stílusok>
 <Piros>
  <Vörös>Ez vöröske</Vörös>
  <Bordó>Ez bordó hordó
   <világosbordó>vBordó</világosbordó>
   <sötétbordó>sBordó</sötétbordó>
  </Bordó>
 </Piros>
 <Kék>Ez bizony kék</Kék>
 <Italic>Ez pedig Italic</Italic>
</Stílusok>
```

Néhány péda:

Legyen az aktuális elem a „Piros”.

child::Vörös	A „Piros” elem gyermekei közül a „Vörös” nevűek.
*child::**	A „Piros” elem összes gyermeke.
child::sötétbordó	A „Piros” elem gyermekei közül a „sötétbordó” nevűek. Miután a „Piros” elemnek nincs „sötétbordó” nevű gyermeke (csak unokája) ezért ez egy üres halmaz.
descendant::sötétbordó	A „Piros” elem leszármazottai közül a „sötétbordó” nevűek.
child::node()	A „Piros” elem összes gyermeke.

8.2.3 Feltételes kifejezés

A feltételes kifejezés egy tetszőleges összetett kifejezés, amely a csomópontok körének meghatározására egyéb XPath kifejezéseket, függvényeket, és a csomópontok tartalmát is felhasználhatja. A kifejezés kiértékelése előtt létrejön egy az irány és a típus által meghatározott csomópont halmaz. Ezután az értelmező végig megy a halmaz elemein, és minden egyes elemre kiértékeli a feltételes kifejezést. Ha a kifejezés az adott elemre igaz értékű, akkor a csomópont bekerül az XPath lépés eredmény halmazába.

Egy feltételes kifejezés egy vagy több kifejezésrészből épül fel, amelyek között logikai műveletek teremtenek kapcsolatot. A kifejezéshez a következő műveleti jelek használhatók fel:

- **or** (logikai vagy)
- **and** (logikai és)
- = (egyenlőség vizsgálat), **!=** (nem egyenlőség vizsgálat)
- <= (kisebb vagy egyenlő), < (kisebb), >= (nagyobb vagy egyenlő), > (nagyobb)

Itt a sorrend a művelet rendjét is jelzi (tehát az „or” művelet a legmagasabb rendű). A kifejezés pontos megadásához használható zárójelezés, a kisebb és nagyobb jelek pedig felcserélhetők az <, > kifejezésekkel. A feltételes kifejezés nem kötelező, elhagyása esetén a szögletes zárójelek sem szükségesek.

Egy kifejezésrész lehet valamilyen szöveg, szám, függvényhívás vagy XPath kifejezés.

példák:

//CD[Előadó=”Michael Jackson”]	Kiválasztja azon CD elemeket melyeknek van “Michael Jackson” tartalmú gyermek elemük.
chapter[title]	Kiválasztja az aktuális elem azon chapter nevű gyermekeit, amelyeknek van title nevű gyermeke.
`para[5][@type="warning"]`	Kiválasztja az aktuális elem 5. gyermekét, feltéve, hogy annak van warning tartalmú és type nevű tulajdonsága

8.2.4 Rövidítések

A leggyakrabban használt XPath kifejezéseket a könnyebb fejlesztés érdekében rövidített formában is el lehet érni, ezek a következők:

(semmi)	child::
@	attributes::
. (pont)	self::node()
.. (két pont)	parent::node()
//	/descendant-or-self::node()/

Lássunk egy példa kifejezést: *descendant::sötétbordó/parent::node()*

Ha továbbra is a „Piros” elem az aktuális, akkor ez az XPath elérési-út a „Bordó” elemet jelenti. A kifejezés rövidítve: *.//sötétbordó/..*

A jobb érthetőség kedvéért kicsit jobban kifejtve:

self::node()/descendant-or-self::node()/child::sötétbordó/parent::node()

Megjegyzés: A „//Elemnév” elérési-út kifejtve „/descendant-or-self::node()/child::Elemnév” ami kiválasztja először a teljes XML fát, majd abból az összes „Elemnév” elemet függetlenül annak a helyétől.

8.2.5 Függvények

Csomópont halmaz függvények

last()	Megadja az aktuális halmaz utolsó elemének indexét. (vagyis a halmaz méretét)
position()	Megadja az aktuális elem indexét.
count(halmaz)	Megadja a paraméterként megadott halmaz csomópontjainak számát.
id(objektum)	Az adott azonosítóval rendelkező elemet adja vissza. Csak séma definícióval rendelkező dokumentum esetében működik, hiszen csak ebben az esetben állapítható meg, hogy mi az azonosító.
local-name(halmaz)	Ha az aktuális elem kiterjesztett névvel rendelkezik, akkor a függvény megadja ennek a kiterjesztett névnek a helyi (local) részét.
namespace-uri(halmaz)	Ha az aktuális elem kiterjesztett névvel rendelkezik, akkor a függvény megadja ennek a kiterjesztett névnek a névtér URI részét.
name(halmaz)	Vissza adja az aktuális elem nevét.

String függvények

string(objektum)	A paraméterként megadott objektumot szöveggé alakítja át.
concat(String, String*)	A paraméterként megadott - korlátlan számú - karakterláncot összefűzi.
starts-with(String, String)	Igaz, ha a paraméterként megadott első sztring a második sztringgel kezdődik.
contains(String, String)	Igaz, ha a paraméterként megadott első sztring tartalmazza a második sztringet.
substring-before(String, String)	Vissza adja az első string mindazon részét, ami a második string első stringben való első előfordulása előtt van. Ha az első string nem tartalmazza a második stringet, az eredmény üres string.
substring-after(String, String)	Vissza adja az első string mindazon részét, ami a második string első stringben való első előfordulása után van. Ha az első string nem tartalmazza a második stringet, az eredmény üres string.
substring(string, number, number?)	Az első stringnek az első szám által meghatározott pozíciótól kezdődő a második szám által meghatározott számú karakterét adja vissza. Ha a második szám hiányzik, akkor a visszaadott string az eredeti string végéig tart.
string-length(string?)	A paraméterként megadott, vagy annak hiányában az aktuális elem értékét számmá alakítja.
normalize-space(string?)	A paraméterként megadott stringből, vagy annak hiányában az aktuális elemből kiveszi a fölösleges kihagyás jellegű karaktereket.
translate()	Kis-nagy betű konverzióra szolgáló függvény.

Logikai függvények

boolean(objektum)	A paraméterként megadott objektumot logikai értékké alakítja át.
not(boolean)	A paraméterként megadott logikai értéknek az ellentettjét adja vissza.
true()	Mindig igaz értéket ad vissza. Nincs paramétere.
false()	Mindig hamis értéket ad vissza. Nincs paramétere.
lang()	Megadja, hogy az adott csomópont azonos nyelvű-e a paraméterként megadott nyelvvel. Egy csomópont (ez a továbbiakban minden leszármazottra öröklődik) nyelvét az xml:lang tulajdonság segítségével lehet megadni. Ritkán használják.

Számtani függvények

number(objektum)	A paraméterként megadott objektumot számmá alakítja át.
sum(halmaz)	Megadja a paraméterként megadott halmaz számértékeinek összegét.
floor(number)	A paraméterként megadott számot lefelé kerekíti.
ceiling(number)	A paraméterként megadott számot felfelé kerekíti.
round(number)	A paraméterként megadott számot kerekíti.

8.3 Műveletek

Ha egy XPath kifejezés egy elérési-út, akkor eredményként csomópontok egy halmaza adódik. Az XPath azonban lehetőséget ad további műveletek végzésére, mely műveletek paraméterei lehetnek elérési-utak által meghatározott csomópont halmazok, szövegek, számok, a műveletek pedig lehetnek halmaz, matematikai, szöveg vagy logikai műveletek.

8.3.1 Halmaz műveletek

Az egyetlen halmaz művelet az unio, amellyel halmazokat lehet egyesíteni, ennek jele a „|”.

8.3.2 Logikai műveletek.

Logikai műveletek értéke mindig vagy igaz, vagy hamis. Logikai műveletekhez a korábban már megismert logikai műveleti jelek használhatók. Logikai műveletek értelmezhetők logikai értékek, számok, szövegek és csomópont halmazok között is, a művelet mindig két operandus között értelmezhető.

Logikai vagy (or) művelet értéke igaz, ha a művelet bármely operandusa igaz. Amennyiben a bal oldali operandus értéke igaz, a jobb oldali operandus értéke nem kerül kiszámításra.

Logikai és (and) művelet értéke igaz, ha a művelet mindkét operandusa igaz. Amennyiben a bal oldali operandus értéke hamis, a jobb oldali operandus értéke nem kerül kiszámításra.

Amennyiben mindkét operandus halmaz, akkor a műveletek úgy hajtódnak végre, hogy a halmazok elemei szöveggé alakítódnak. Ekkor a két halmazon páronként értelmeződnek a relációk. Ha a reláció az egyenlőség, akkor az eredmény akkor igaz, ha a két halmaz megegyezik, ha a reláció a nem egyenlő, akkor a reláció akkor igaz, ha a párok közül akad legalább egy, amelyek nem egyenlők. Ha az egyik operandus Op1(„hello”, „8”, „15”) a másik pedig Op2(„hello”, „8”, „15”), akkor Op1=Op2 igaz értékű. Ha Op3(„8”), akkor Op1=Op3 hamis.

Amennyiben az egyik operandus csomópont halmaz, a másik pedig szám, akkor a művelet értéke akkor és csak akkor igaz, ha a halmaz egy elemű, ez egy szám és a reláció ezen szám és a második operandus között igaz. Op1=8 tehát nem igaz, ugyanakkor Op3=8 igaz.

Amennyiben az egyik operandus csomópont halmaz, a másik pedig szöveg, akkor a művelet értéke akkor és csak akkor igaz, ha a halmaz egy elemű és ennek az elemnek az elemnek a megfelelő relációja a második oparandussal igaz.

Amennyiben az egyik operandus sem halmaz és a művelet az egyenlőség vagy a nem egyenlőség, akkor mindkét operandus megfelelő típusúvá alakítódik, és végrehajtódik a művelet.

Amennyiben az egyik operandus sem halmaz és a művelet kisebb, kisebb egyenlő, nagyobb vagy nagyobb egyenlő, akkor mindkét operandus számmá alakítódik, és végrehajtódik a művelet.

8.3.3 Számtani műveletek

Az XPath dupla pontosságú 64 bites lebegőpontos számokat kezel. Az érték tartományba bele értendők a negatív és a pozitív számok, a negatív és a pozitív végtelen, a negatív és a pozitív nulla, és egy speciális nem szám (Not a Number – NaN) érték.

A számtani operátorok a művelet végrehajtása előtt elvégzik az operandusokon a number() függvény által megvalósított szám konverziót (Vagyis minden paraméter számmá vagy NaN értékké alakul).

Operátorok:

+	Összeadás
-	Kivonás
*	Szorzás
div	Lebegőpontos osztás
mod	Maradékos osztás maradéka

8.4 XPath példák

```
//CD
//CD/Artist
//CD[Artist=”Queen”]
count(//CD)
sum(//CD/sold[../Artist=”Queen”])
```

9. XSL

Az XSL egy olyan XML alapú sablont jelent, ami az XML dokumentum megjelenítésére használható. Lehetőséget ad a dokumentum adatok transzformációjára – ami jelentheti bizonyos elemek elhagyását épp úgy, mint elemek sorrendjének megváltoztatását, a dokumentum struktúrájának megváltoztatását, közvetetten meghatározható adatok, például kategóriák számának meghatározását, stb. –, az adatok tetszőleges pozicionálására, illetve tetszőleges stílus elemek meghatározására.

Az XSL valójában két technológia. Az egyik egy hagyományos stíluslap lehetőségeit kínálja megjelenésbeli stílusok, formák, helyzetek meghatározására, ez az XSLFO (XSL Formatting Objects), ehhez az xmlns:fo="http://www.w3.org/1999/XSL/Format" névtér tartozik. Képességeiben, funkcionalitásában erősen hasonlít a CSS-re, bár nyelvi elemeiben természetesen teljesen más - nyilván hiszen ez egy XML dokumentum is egyben. A másik igazából nem is stíluslap, sokkal inkább sajátos programozási módszer, nyelv, beépített vezérlési struktúrákkal, nyelvi elemekkel, paraméterekkel, változókkal, csak éppen az XML szabályainak megfelelően egy XML dokumentumként leírva. Ez utóbbi az XSLT (XSL Transformation) a névtere pedig: xmlns:xsl="http://www.w3.org/1999/XSL/Transform". Fontos, hogy az XSLT-vel kapcsolatban több olyan

példát is lehet találni, ahol a névtér nem ez (például: xmlns:xsl="http://www.w3.org/TR/WD-xsl"). Ez azért van, mert korábban más volt a névtér azonosító. A lényeges az, hogy a transzformációs motorok a névtér nevét használják az adott változat azonosításához (valamint a verzió számot), vagyis, egy régebbi névtér név használata azt eredményezheti, hogy az egyébként tökéletes XSLT-t nem hajtja végre az értelmező. Ugyanez a helyzet, ha hibás a névtér neve.

Példa XSL dokumentumra, amely transzformációt és formázást is tartalmaz:

```
<xsl:template match="chapter">
 <fo:flow>
  <xsl:apply-templates/>
 </fo:flow>
</xsl:template>

<xsl:template match="chapter/title">
 <fo:block font-size="18pt" font-weight="bold" text-align="centered">
  <xsl:apply-templates/>
 </fo:block>
</xsl:template>

<xsl:template match="para">
 <fo:block font-size="12pt" space-before="1pc" text-align="justified">
   <xsl:apply-templates/>
 </fo:block>
</xsl:template>

<xsl:template match="emphasis">
 <fo:inline font-style="italic">
  <xsl:apply-templates/>
 </fo:inline>
</xsl:template>
```

Látható, hogy ez egy közönséges XML dokumentum, amelyben "template", "flow", "apply-templates" és számos egyéb elem található, amelyek ugyanakkor egy értelmező számára lényeges jelentéssel bírnak. Az is látható, hogy egy XSL dokumentumon belül az XSLT és az XSLFO elemek keveredhetnek, egészen pontosan kiegészítik egymást. Mi most a két technológiát a könnyebb érthetőség kedvéért külön tárgyaljuk.

9.1 XSLT

Egy XSLT dokumentum feladata a transzformáció. A meglévő XML dokumentumot, annak adatait használja fel, hogy egy akár az eredetitől teljesen különböző adathalmazt és adat struktúrát állítson elő. Az, hogy azután ezekből az adatokból egy másik XML dokumentum, vagy egy HTML szerű megjelenés jön-e létre az tulajdonképpen mellékes.

9.1.1 Felépítés

Egy XSL dokumentum (itt mindegy, hogy XSLT vagy XSLFO) miután egy XML dokumentum is egyben természetesen az *xml* vezérlési utasítással és a verzió megadásával kezdődik, valamint a dokumentum gyökere a *stylesheet* elem, ahol rögtön megadásra kerül a használt névtér is, valamint meg kell adni a verziót.

Egy XSLT dokumentum alap eleme sablon. Sablonokat definiálhatunk egy XML dokumentum valamely csomópontjához - bármely csomópontjához, sőt akár minden csomópontjához -, hogy az azon a ponton található adatokra, illetve az onnan elérhető gyermek elemekre vonatkozó manipulációt végre hajthassuk.

Ez a manipuláció tetszőleges átalakítás lehet, de mi most példáinkban az egyszerűség kedvéért egy HTML dokumentumot állítunk elő. Ez azért jó, mert így egy böngészőben egyszerűen nyomon követhetjük az eredményt. A pontosság kedvéért nem HTML, hanem XHTML dokumentumot készítünk - ez nyelvi elemeit tekintve teljesen azonos a HTML 4.01 specifikációban adottakkal, azonban XML dokumentum, ezért érvényesek rá az XML szigorú szabályai (tehát például nincs <BR>, csak <BR/>, stb.).

Egy XSLT dokumentum tehát sablonokból, de mindenképpen legalább egy sablonból áll, melyet a *template* elem segítségével definiálhatunk.

Példa egyszerű XSLT lapra (pelda_01.xsl):

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
  <html>
  <body>
     <xsl:value-of select="."/>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Ez a lap egyetlen sablont tartalmaz. A sablonok definíciójuk szerint abban térnek el egymástól, hogy mely csomóponthoz – vagy mely csomópontokhoz – tartoznak. (Itt a csomópont az eredeti, transzformálni kívánt XML dokumentum csomópontja.) Minden sablonhoz tartozik egy a csomópontokat meghatározó csomópont leírás. Amelyik sablonhoz egyetlen csomópont sem tartozik (mert például olyan feltételes kifejezés van megadva, amelyet egyetlen csomópont sem elégít ki a jelenlegi XML fában) az nem is fog lefutni.

A csomópont megadásához a match paramétert kell megadni, ahol az érték a sablonhoz tartozó csomópontokat jelöli ki. Létezik egy szabvány egy XML dokumentum csomópontjainak megadására, ez az XPath – az XSLT esetén is ezt kell használni.

A legegyszerűbb, a fájlrendszerben jól ismert megadási mód (elvégre a fájl rendszer is egy fa struktúrát reprezentál). A "/" jelenti a gyökér elemet "." az aktuális elemet, a "Tárgy/Autó" pedig a Tárgy elem Autó gyermek elemét jelenti. Ez a megadási mód azonban csak egy az XPath lehetőségei közül. Megadható, például egy elem, bárhol van is a dokumentum fában (//Autó), sőt ekkor minden Autó elemet kijelöl ez a megjelölés. Feltételt is megadhatunk egy XPath kijelölésben. A "//Meccs[HazaiCsapat = 'Fradi']" kifejezés kijelöl minden olyan meccs elemet, melynek van olyan HazaiCsapat gyermek eleme, amelynek az értéke 'Fradi' (például kijelöli az összes olyan meccset, ahol a Fradi otthon játszik.). Az XPath elég széles lehetőségi körrel rendelkezik, erről később lesz szó.

A fent látható példában egy további elem a value-of is megtalálható. Ez valamely érték beillesztésére használatos. A konkrét érték a select paraméter segítségével adható meg, ez ismét egy XPath kifejezés lehet, ami jelen esetben az aktuális (vagyis a gyökér elem) tartalmát jelenti. Minthogy a value-of elem esetén a jelenlegi példában nincs szükség bezáró párra, ezért itt ezt természetesen a fenti módon jelezni kell.

Legyen a fenti XSLT-hez tartozó XML a következő (pelda_01.xml):

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="pelda 01.xsl"?>
<Stílusok>Ez a lap stílusokról szól.</Stílusok>
```

A megjelenítéshez persze itt is (akár csak a CSS stíluslapok esetén) meg kell adni, hogy milyen lapot használjon a böngésző. Erre a fenti vezérlési utasítás szolgál. Más módja is van egy XML dokumentum és egy XSLT lap összerendelésének, de ez a legegyszerűbb ezért most ezt használjuk. Ekkor az eredmény valahogy így néz ki:

```
Ez a lap stílusokról szól.
```

9.1.2 Sablonok alkalmazása

Természetesen egy XSLT lap tetszőleges számú sablont tartalmazhat. Több sablon használatának az értelme az, hogy így minden csomóponthoz külön megmondhatjuk, hogy az értelmező hogyan dolgozza fel azt. Elvileg egyetlen sablon használatával is meg lehet oldani mindent, de több sablon használata strukturáltságot biztosít. (Egy picit olyan mint függvényeket és eljárásokat használni.)

Legyen a példa XML a következő (pelda_02.xml):

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="pelda_02.xsl"?>
<Stílusok>
 <Piros>
  <Vörös>Ez vöröske</Vörös>
  <Bordó>Ez bordó hordó</Bordó>
 </Piros>
 <Kék>Ez bizony kék</Kék>
 <Dőlt>Ez pedig dolt</Dőlt>
</Stílusok>
```

Készítsünk most egy XSLT lapot, amely minden csomóponthoz rendel egy sablont, és annak a tartalmát értelemszerűen jeleníti meg (pelda_02.xsl):

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:template match="Vörös">
    <font Color="#FFAA00">
       <xsl:value-of select="."/>
    </font>
  </xsl:template>

  <xsl:template match="Bordó">
    <font Color="#900000">
       <xsl:value-of select="."/>
    </font>
  </xsl:template>

  <xsl:template match="Kék">
    <font Color="#0000A0">
       <xsl:value-of select="."/>
    </font>
  </xsl:template>

  <xsl:template match="Dőlt">
    <i>
       <xsl:value-of select="."/>
    </i>
  </xsl:template>

</xsl:stylesheet>
```

Ekkor a képernyőn megjelenik az XML dokumentum összes tartalmi eleme (az összes szöveg) egymás után, úgy minden felirat a megadottnak megfelelő színű. Felmerül a kérdés, mi történik, ha nincs minden csomóponthoz sablon készítve. Ennek megvilágítására szolgál a következő példa. Az XML dokumentum az előzővel azonos tartalmú (pelda_03.xml), az XSLT lap pedig a következő (pelda_03.xsl):

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:template match="Piros">
    <font Color="#FF0000">Piros szöveg
    </font>
  </xsl:template>

  <xsl:template match="Vörös">
    <font Color="#FFAA00">
       <xsl:value-of select="."/>
    </font>
  </xsl:template>

  <xsl:template match="Borddó">  <!-- Szándékosan elrontva -->
    <font Color="#900000">
       <xsl:value-of select="."/>
    </font>
  </xsl:template>

</xsl:stylesheet>
```

Ekkor megjelenik a "Piros szöveg" felirat, ami várható is, hiszen a "piros" elemre vonatkozó sablon ezt írja elő. Figyeljük meg, hogy egy adott csomóponthoz tartozó sablonnak nem feltétlenül kell törődnie az adott csomópontból elérhető adatokkal. Meglepő ugyanakkor, hogy a "Vörös" elem tartalma meg sem jelent, holott létezik hozzá sablon. Ráadásul a „Kék” és a „Bordó elem tartalma megjelent, holott ezekhez nincs is sablon rendelve. Mindez persze nem véletlen.

Amikor az értelmező elkezdi feldolgozni az XML dokumentumot, akkor először **fog egy csomópontot, majd keres hozzá egy sablont**. Ha az adott csomóponthoz **nem talál** sablont, **akkor tovább lép** az adott elem **gyermek elemére**, és ahhoz keres sablont. Ha az adott elemnek nincs gyermeke, akkor az adott elem testvére következik (és persze annak a gyermeke), majd ha a testvérek is elfogytak, akkor visszalép egyet, és a szülő testvérével folytatja. Mindez addig tart, amíg bejárja az egész XML fát. Ha valamely elemhez az értelmező talál egy sablont, akkor végre hajtja az adott sablont, de ha a sablon végrehajtása befejeződött, akkor az értelmező az adott elem gyermekelemeire nem lép már tovább, mert feltételezi, hogy azt a szülőhöz tartozó sablon már lekezelte (erre van mód, és erről még lesz szó). Emiatt történik, hogy bár a "Vörös" elemhez létezik sablon, azzal az értelmező már nem foglalkozik.

Ha egy elemhez nem talált sablont (sem annak valamely felmenőjéhez), akkor az értelmező adott elem tartalmát a kimenetre másolja - ez a magyarázata annak, hogy bár sem a "Kék" elemhez sem a "Bordó" elemhez nincs

sablon, az eredményben mégis megtaláljuk a tartalmukat - és természetesen az sem véletlen, hogy ekkor teljesen formázatlanok.

Az XSLT végig járja az XML fát, végig megy annak minden elemén, majd az adott aktuális elemhez veszi a megfelelő sablont, és végre hajtja azt. Fontos, hogy ha egy csomóponthoz talál megfelelő sablont, akkor annak a csomópontnak a gyermekein már nem megy végig, nem keres hozzájuk sablont. Egészen pontosan, csak akkor foglalkozik velük, ha erre az adott szülő sablon külön kéri. Másképpen, az értelmező úgy tekinti, hogy egy sablon végrehajtásával – hacsak az másképpen nem rendelkezik – a gyermek elemek transzformációja is befejezettnek tekinthető.

apply-templates

Az adott sablon az értelmezőt az *apply-templates* elem segítségével utasíthatja arra, hogy gyermek elemek transzformációját is végezze el (példa 04). Alapértelmezés szerint minden gyermek elemen végig megy (az egész XML fán, ami az adott elemből indul ki), de a *select* paraméter segítségével (példa 05) megadható egy XPath kifejezés, hogy csak az annak megfelelő csomópontokon menjen végig. Tulajdonképpen a kifejezés egy gyermek-csomópont készletet jelöl ki, amelyen azután végig megy az értelmező.

Az *apply-templates* elem a sablonon belül bárhol elhelyezhető. Ha közvetlenül a sablon elején található, akkor az értelmező tulajdonképpen először a gyermek elemeken megy végig, és csak ezt követően hajtja végre az adott sablont - annak a végrehajtását addig felfüggeszti. Ha a végén szerepel, akkor először végre hajtja a sablont, majd veszi a gyerekeket. Ha az *apply-templates* elem a sablon belsejében található (példa 06), akkor először végre hajtódik a sablon eleje, majd a gyermek elemekre kerül a vezérlés, végül a gyermek elemek végrehajtását követően befejeződik a sablon végrehajtása.

Ahogy már az előbb is szerepelt, az értelmező a kimenetére másolja mindazon csomópontok tartalmát, amelyekhez - sem valamely felmenőjéhez - nem tartozik sablon. Ez a legtöbb esetben nem kívánt viselkedés, ezért majdnem minden XSLT lap egy a gyökér elemhez tartozó sablonnal kezdődik, akkor ugyanis az előbbi mellékhatás kiküszöbölődik, miközben az *apply-templates* elem segítségével a további sablonok végrehajtása is vezérelhető.

Példa sablonok használatára - az XML még mindig a 2. példánál használt (pelda_06.xsl).

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:template match="/">
    <H1>Színek</H1><HR/>
     <xsl:apply-templates/>
    <HR/>
  </xsl:template>

  <xsl:template match="Piros">
    <font Color="#FF0000">Piros szöveg</font><BR/>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="Vörös">
    <DD/><font Color="#FFAA00"><xsl:value-of select="."/></font>
  </xsl:template>

  <xsl:template match="Bordó">
    <DD/><font Color="#900000"><xsl:value-of select="."/></font>
  </xsl:template>

  <xsl:template match="Kék">
    <DD/><font Color="#0000A0"><xsl:value-of select="."/></font>
  </xsl:template>

  <xsl:template match="Italic">
    <DD/><i><xsl:value-of select="."/></i>
  </xsl:template>

</xsl:stylesheet>
```

value-of

Miután az XSLT az XML dokumentum transzformációját végzi, elengedhetetlen, a dokumentumból következő értékeknek (csomópontok értékei, paraméter értékek, értékek száma, összege, csomópontok neve, stb.) a transzformáció kimenetén való megjelenítése (természetesen a kívánt módon átalakítva) erre a *value-of* elem használható fel. Az elem *select* paramétere segítségével választható ki az a tulajdonképpeni érték amely megjelenítésére szükség van. Ez lehet egy XPath kifejezés - például a már látott "." amely az aktuális elemet, így annak értékét jelöli ki -, valamely XSLT függvény, illetőleg ezek valamilyen keveréke.

példa:

```
<xsl:value-of select="count(*)"/>
```

Ez a példa megszámolja az aktuális elem gyermek elemeinek a számát (Lásd: példa 07).

for-each

ciklus

sort

under construction

variable

csak inicializálni lehet

under construction

if

feltételes elágazás

under construction

choose

feltételes elágazás

when

otherwise

under construction

9.2 XSLFO

under construction

10. Programozás

Az XML-t kezelő alkalmazásokat fejlesztők dolgát megkönnyíti, hogy rögtön kétfajta értelmező is rendelkezésre áll XML dokumentumok programból történő kezeléséhet, értelmezéséhez. Mindkettő elrejti a programozó elől az XML szintaktikáját, így a fejlesztőnek csak a tartalom helyes kezeléséről kell gondoskodnia.

10.1 SAX

A SAX (Simple API for XML) egy eseményvezérelt értelmező. A dokumentumot folyamatosan olvasva a felbukkanó elemeket és adatokat egy-egy metódushívássá konvertálja, a konkrét értelmezést pedig rábízza az alkalmazás készítőjére, akinek feladata az, hogy az értelmező osztályt kiterjesztve megírja az egyes metódusok törzsét, és saját céljai szerint értelmezze az adatokat.

10.2 XML DOM

A DOM (Document Object Model) értelmező az XML dokumentumot egy az egyben beolvassa, és egy objektumokból épített fát állít elő belőle, amely leképezi a dokumentum struktúráját. Ezek után tetszés szerinti elem bármely adata elérhető. Az XML DOM számos programozási nyelvben használható a C++-tól a JavaScriptig ezért kiválóan alkalmazható például kliens oldali web alkalmazások programozására.

Példa XML DOM használatára Javascriptben:

```
<html>
<body>
<script type="text/javascript">
// Load XML
var xml = new ActiveXObject("Microsoft.XMLDOM")
xml.async = false
xml.load("cd_catalog.xml")

// Load the XSL
var xsl = new ActiveXObject("Microsoft.XMLDOM")
xsl.async = false
xsl.load("cd_catalog.xsl")

// Transform
document.write(xml.transformNode(xsl))
</script>
</body>
</html>
```

A program működése egy XMLDOM objekjum létrehozásával kezdődik, amihez hozzá kell rendelni a kívánt XML állományt. Ezután tetszőleges módon használható az állomány. A jelent példa létrehoz egy újabb XML objektumot, amihez egy XSL sablont rendel, majd megjeleníti az XML állományt az XSL sablon segítségével. Ez a program egy lehetőség tetszőleges adat és sablon állományok összerendelésére.

11. Az XML egyéb alkalmazásai

Az XML azon túl, hogy adat leíró nyelv, és lehetővé teszi adatok szabványos kommunikációját, számos alkalmazás leírónyelve is egyben. Ilyen alkalmazás például a már megismert XSL, ami egy megjelenés leíró sablont definiáló leíró nyelv, amely azonban egyszersmind XML alkalmazás.

11.1 XHTML

Az XHTML újabb jó példa az XML alkalmazásokra. Olyan XML alkalmazás amely elem neveit és célját illetően teljesen megegyezik a HTML 4.01 szabványban rögzített HTML nevekkel és képességekkel, ugyanakkor vonatkoznak rá az XML alkalmazások szigorú kritériumai.

11.2 XML Schema

Az XML Schema ismét egy XML alkalmazási nyelv amelynek a célja megegyezik a DTD céljával, tehát típus definíciók leírására szolgál. Kifejezetten az XML-hez készült, a DTD-nél egyszerűbb, ráadásul kínálja azt az előnyt, hogy XML szintaktikájú, tehát használatához (a DTD-vel ellentétben) nincs szükség új szintaxis megtanulására. Ma az XML alkalmazások típus definícióihoz a szabvány már az XML Schema-t ajánlja.

11.3 EbXML

Az ebXML egy szabványos, az egymással kompatibilis, egymással változtatás nélkül kommunikálni akaró eBusiness alkalmazásokhoz kifejlesztett XML alkalmazás nyelv.

12. XML-el kapcsolatos anyagok a neten

http://www.vbxml.com/
Itt számos webes technológiáról van tutorial és referencia. Elég jó és jól érthető anyagaik vannak, részletesek, és a példák is működnek.

http://www.zvon.org/
Szintén nagyszerű és részletes tutorialok, és referenciák találhatók itt, bár a példák többsége nem működik az Internet Explorerrel, csak más transzformációs motorral.

http://www.w3schools.com/
Könnyen érthető, tutorialok XML-ről és a kapcsolódó technológiákról, nem nagyon mély, de kezdőnek ideális.

http://msdn.microsoft.com/xml/default.asp
XML-el kapcsolatos információk a Microsofttól. Külön érdekes a download szekció, ahol a web development rész alatt tálalhatók az XML-hez tartozó letölthető dolgok, például XML parserek, és javítások.

http://www-106.ibm.com/developerworks/xml/
IBM XML fejlesztői információk és más erőforrások. Külön ajánlom a http://www-106.ibm.com/developerworks/xml/library/hands-on-xsl/index.html?dwzone=xml címen tálalható leírást, az XML transzformációról.

http://www.tech.irt.org/articles/xml.htm
Itt cikkek vannak XML-ről és kapcsolódó technológiákról.

http://technet.oracle.com/tech/xml/
Oracle XML központ

http://www.alphaworks.ibm.com/
Itt elsősorban programok vannak jellemzően IBM fejlesztésűek új technológiákhoz, általában java nyelven. XML-hez is vannak dolgok.

http://www.xslt.com/
XSLT-hez kapcsolódó mindenfele dolgok - transzformációs motorok, tutorialok, segéd programok, stb.

http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm
A netscape Javascript referencia leírása.

http://www.w3.org/
Webes technológia és szabvány leírások. Száraz, nehezen emészthető, hosszú, de mindenre kiterjed.