XmlGessünk 15: XSLT

Az egyik legellentmondásosabb és mégis nagyon sűrűn használt xml technológia az XSLT. Barátkozzunk meg vele!

X aknák

Tisztázzunk néhány X fogalmat. Az XML az Extensible Markup Language rövidítése. Ez definiálja egy XML dokumentum struktúráját [1]. Az XML dokumentumok által leírt információk struktúráját, azaz egy XML doksi tényleges információtartalmának szerkezetét az XML Infoset szabvány írja le [2]. Ebben a kacsacsőröktől, idézőjelektől és minden egyéb szintaktikai sallangtól mentesen definiálnak egy objektummodellt, amely az xml dokumentumok logikai adatstruktúráját írják le. A jövőbeli szabványok már valószínűleg erre fognak támaszkodni, és végre elszakadhatunk a konkrét szöveges formátumtól.
Az XSL néven jelzett szabvány (Extensible Stylesheet Language) valójában három további részből áll [3]. Az elsőt nevezik XSLT-nek (XSL Transformations) [4], ez xml dokumentumok transzformációjára kidolgozott nyelv. Ez lesz a fő tárgyalási irányunk. Az XSLT-ben szükség van a forrásdokumentum különböző részeinek kijelölésére, erre dolgozták ki az XPath [5] szabványt.
Amikor XSL transzformációkkal foglalkozunk, leginkább az XSLT és az XPath szabványokat használjuk. Az XSL szabvány harmadik részével, a Formatting Objects-nek nevezett általánosított formátumleírással nem foglalkozunk, mivel a gyakorlatban se nagyon terjed el.

Mire használható az XSL(T)?

Az első időkben az XSL kidolgozásának célja kifejezetten a megjelenítés támogatása, azaz xml dokumentumok valamilyen médiumon keresztüli megjelenítése, formázása volt. A mai webes munkákban továbbra is használjuk ezt az aspektusát, habár (általában) a Formatting Objects kihagyásával közvetlenül HTML kimenetet állítunk elő.
Vannak xml alapú grafikus szabványok is, így semmi akadálya, hogy például egy adatbázis kimeneteként előállt xml dokumentumból XSLT segítségével röptében képet generáljuk.
Az XSLT azonban hamarosan önálló szabvánnyá nőtte ki magát, és elkezdték használni különböző xml sémák közötti átjárásra. Tehát nem adatok megjelenítése, hanem az adatstruktúra átalakítása lett a legfőbb csapásirány.

XSLT alapok

Az XSLT egy szabályalapú, deklaratív programozási nyelv. Szabályalapú, mert template-ekkel (sablonokkal) leírhatjuk, hogy a forrásdokumentum melyik részét mivé szeretnénk transzformálni. Csak leírjuk, deklaráljuk, hogy miből mi lesz, a transzformáció tényleges folyamát a forrásadatok és az XSLT transzformációt végző motor szabja meg. A sablonok végrehajtási sorrendje nem determinálható az XSLT írásakor, csak futásidőben derül ki. Emiatt ha a sablonok között volnának egymásra hatások, a transzformáció kimenete sokszor nem az lenne, amire számítunk. Ezért az XSLT-t szándékosan mellékhatás-mentesre írták meg, ami olyan, sokszor bosszantó korlátozásokban jelenik meg, mint hogy például a globális változóknak csak egyszer lehet értéket adni.
A deklaratív jelleg igencsak csípi a szemét annak, aki világ életében a C, VB vagy egyéb procedurális nyelveken nőtt fel. Első ránézésre furcsa lesz, de minden korlátozása ellenére az XSLT életképes technológia, amibe úgy tűnik érdemes befektetni.

Helló világ

Első példánkban az alábbi dokumentum lesz a kiindulásunk:

```
<?xml version='1.0' encoding="ISO8859-2" ?>
<hello>
   <pajti>Soci</pajti>
   <duma>Üdvözlet a világnak!</duma>
</hello>
```

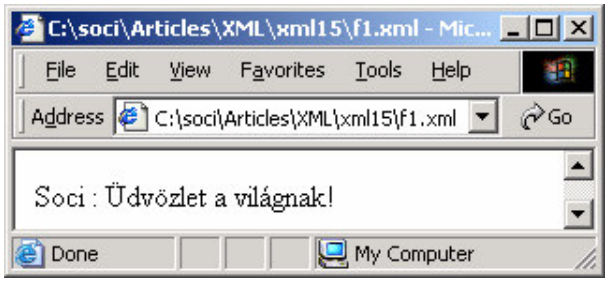
Az alábbi transzformációt fogjuk rá alkalmazni:

```
1 <?xml version="1.0" encoding="ISO8859-2" ?>
2 <xsl:stylesheet version='1.0'
  ↳xmlns:xsl='http://www.w3.org/1999/
  ↳XSL/Transform'>
3   <xsl:template match="/">
4     <xsl:value-of select="hello/pajti" />
5     :
6     <xsl:value-of select="hello/duma" />
7   </xsl:template>
8 </xsl:stylesheet>
```

A teszteléshez első körben az Internet Explorert fogjuk használni, amely képes xslt transzformáció közvetlen végrehajtására, ha azt hozzáláncoljuk a forrásdokumentumhoz. Ehhez az xml forrásunkba be kell szúrni egy vezérlőutasítást az xml deklaráció és a gyökérelem közé:

```
<?xml version='1.0' encoding="ISO8859-2" ?>
<?xml-stylesheet type="text/xsl" href="t1.xsl" ?>
<hello>
```

Ezek után már csak be kell töltenünk a forrásdokumentumot az IE-be:



Az Explorer felolvassa a forrásdokumentumit, betölti a hozzá tartozó transzformációt, végrehatja azt a forráson, és mi már a transzformáció kimenetét láthatjuk a böngészőablakban.
Néhány szó a szükséges szoftverekről. A transzformációk helyes működéséhez legyen fenn a gépen az MSXML3 SP2 (vagy a legutóbbi) programcsomag is. Van már újabb xml parser csomag is az MSXML4 személyében, azonban az előbbi módon végrehajtott transzformációk ezt nem tudják kihasználni. Csak kézi, kódból végrehajtott transzformációban, explicit 4-es verziójú objektumpéldányok létrehozásával lehet élni az új lehetőségekkel (pl. XSD). Általában érdemes mindkét csomagot felrakni, és kódból az újabb verziót használni - az Explorer meg elvan a 3-assal.
De térjünk vissza a transzformációnkra! Az első sor közönséges xml deklaráció, mivel az XSLT nem más, mint egy xml dialektus. A második sor a transzformáció gyökéreleme, ami stylesheet vagy transform lehet, a kettő egyenértékű. Figyeljük meg, hogy az XSLT nyelvtanához tartozó elemek (a transzformációs nyelvtan) az xsl prefixel ellátott névtérbe vannak rendezve. A prefix neve tetszőleges, a transzformációt a hosszú URI azonosítja. Fontos, hogy az URI-t betűről-betűre pontosan írjuk le, különben nem fog működni a transzformáció. Az elírás legbiztosabb jele, ha a transzformáció kimenete maga az XSLT tartalma.
Az összes, kimenetet előállító parancsot xsl:template elemek között helyezzük el, ez a transzformáció egysége, blokkja. A match attribútumban kell megadni, hogy a sablon a bemeneti dokumentum mely részét dolgozza fel. Itt egy XPath kifejezést vár az XSLT processzor. A / a teljes dokumentumot jelenti. Egy normál XSLT-ben általában több sablon is helyet kap, mindegyik más match attribútummal. A match="/" sablon kiemelt jelentőségű, mert ő fut le mindig először. Tekinthető a feldolgozás belépési pontjának, ő a main metódus.
A forrásdokumentumból az xsl:value-of elem segítségével írathatunk ki részeket a kimenetbe. A select attribútumban kell megadni a kiíratandó XPath kifejezést. A hello/pajti egy relatív XPath kifejezés, amely abszolút eléréssel így néz ki: /hello/pajti. A sablonok esetén értelmezhető egy XPath útvonal, amit éppen feldolgoz az adott sablon. Ezt hívják Current Contextnek. Esetünkben ez a /, így adódik ki a relatív utunk teljes alakja.
Az ötödik sorban szereplő kettőspont neve string literal. Nyugodtan vegyíthetünk szövegeket vagy akár xml elemeket is a generált kimenetbe, ezek egyszerűen beleszövődnek a dinamikusan generált tartalomba.
Tegyük fel, hogy a nevet kissé bonyolultabban akarjuk megformázni, például vastag betűvel akarjuk szedni. Ehhez html elemek közé kell rakni a value-of kimenetét. Ezt megtehetjük a gyökérsablonban is, de a könnyebb olvashatóság miatt tegyük ezt ki egy újabb sablonba:

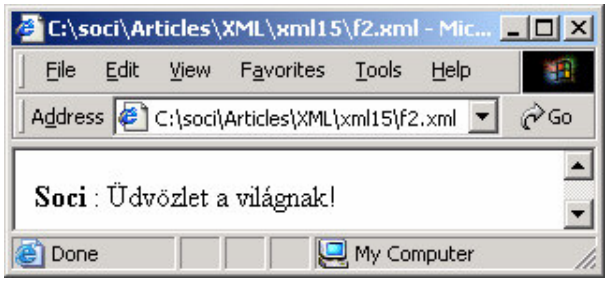
```
  <xsl:template match="pajti">
    <b><xsl:value-of select="." /></b>
  </xsl:template>
```

Látható, hogy ez a sablon a „pajti" nevű elemeket dolgozza fel. Kiíratjuk az Current Context által meghatározott elem tartalmát, amire XPath-ban a „."-tal lehet hivatkozni (hasonlóan a fájlrendszer-könyvárakhoz).
A példát kipróbálva azonban semmi változást nem fogunk tapasztalni. Azért nem, mert az új sablonunk nem fog „csak úgy" elindulni. Tehát az XSLT nem úgy dolgozza fel a forrást, hogy végigfut a forrásdoksin, és minden egyes node-ra megnézi, hogy van-e egyező sablon. Nem, nem így működik. Megnézi, hogy van-e gyökérsablon, ha van, azt meghívja. Ha a gyökérsablon át akarja adni a vezérlést további sablonoknak, azt explicit meg kell tennie! Erre szolgál az xsl:apply-templates elem:

```
  <xsl:template match="/">
    <xsl:apply-templates select="hello/pajti" />
```

Az apply-templates select attribútumában újfent egy XPath kifejezést kell megadnunk. Ez kiválasztja a hello elem pajti nevű gyermekelemét. Egy olyan node halmaz jön létre a memóriában, amiben ez az egyetlen pajti elem lesz. Ezek után az apply-templates úgymond felajánlja: „ki az a sablon, aki tud valamit kezdeni ezzel a node halmazzal, ami egy pajti elemből áll?"

Az új sablonunk boldogan jelentkezik, így a vezérlés átkerül hozzá. Ebben a sablonban a Corrent Context már az egyetlen pajti elemünk lesz, ezért nem kell már a value-of-nak tovább navigálni. Látható, hogy ebben az esetben nem html formázó elemet is generálunk a kimenetbe, aminek a hatása azonnal érzékelhető lesz:



Mi történik, ha a forrásban több, mint egy pajti elem van?

```
<hello>
   <pajti>Soci</pajti>
   <pajti>Betti</pajti>
   <pajti>Fifi</pajti>
   <duma>Üdvözlet a világnak!</duma>
</hello>
```

Ha az előbbi működési módra visszagondolunk, az apply-templates most egy három pajtielemből álló nodehalmazt válogat le, ezért a pajti sablonunk háromszor fog meghívódni:

```
SociBettiFifi : Üdvözlet a világnak!
```

Mindenféle ciklusszervezés nélkül könnyedén végigmehetek tetszőleges számú node-on, és formázottan jeleníthetem meg őket. Ez eléggé furcsa a procedurális technikához szokott fejeknek, hisz ott azzal kezdenénk, hogy írjunk egy ciklust, ami bejárja az elemek halmazát. Ha valakinek ez nagyon hiányzik, megvan rá a lehetőségünk:

```
  <xsl:template match="/">
    <xsl:for-each select="hello/pajti">
      <b><xsl:value-of select="." /></b>
    </xsl:for-each>
```

Most a for-each elem belsejében található tartalom hajtódik végre a select attribútumban kijelölt node-okra. Kimenete azonosa az előbbi példáéval.
Mind az apply-templates, mind a for-each dokumentum sorrendben generálja le a node listát, így a kimenet ennek megfelelő lesz. Gyakori, hogy ez nekünk nem jó, például névsorban szeretnénk látni a pajtikat. Mi sem egyszerűbb:

```
    <xsl:for-each select="hello/pajti">
      <xsl:sort select="."/>
      <b><xsl:value-of select="." /></b>
    </xsl:for-each>
Kimenet:
BettiFifiSoci : Üdvözlet a világnak!
```

Beraktunk egy sort nevű XSLT elemet a ciklus elejére, és a select attribútumában megadjuk azt az elemet, amire rendezni kell a bejárandó node halmazt. Esetünkben ez a pajti elem tartalma, amire a „.” hivatkozik (a template mellett a for-each is változtatja a Current Contextet, így a „.” mindig az aktuálisan bejárt elemet adja vissza).
Az előbbi rendezésben a rendezendő adat típusa string volt. Ha például az alábbi példát

```
...
<hello>
   <pajti kor="67">Soci</pajti>
   <pajti kor="21">Betti</pajti>
   <pajti kor="4">Fifi</pajti>
</hello>
```

a kor attribútumot, mint számot értelmezve akarjuk rendeztetni, a következőképpen kell a sortot paraméterezni:

```
    <xsl:for-each select="hello/pajti">
      <xsl:sort select="@kor" data-type="number"/>
```

```
      <b><xsl:value-of select="." /></b>,
    </xsl:for-each>
Kimenet:
Fifi, Betti, Soci
```

A @kor a kor nevű attribútumra utal, a data-type pedig arra utasítja az XSLT motort, hogy próbálja meg számmá alaktani a hivatkozott kifejezést, és annak megfelelően rakja sorba az értékeket. Sajnos a sort csak a szöveges és a numerikus típusokat ismeri, például dátumot nem. Ennek egyszerű oka van. Az XSLT és az XPath kidolgozásakor még nem volt általánosan elfogadott XML típusrendszer, amit most az XML Schema (XSD) szabvány testesít meg. Viszont minimális típusokra szükség volt, ezért az XPath szabványban az alábbi típusokat definiálták: szöveg, szám, bool, nodehalmaz, xml-dokumentumtöredék. Semmit dátum vagy egyéb hétköznapi típus! Valószínűleg az XPath 2.0 szabvány már az XSD típusaira épít, és innentől kezdve (XSLT 2.0) a sort is használható lesz bármely sématípusra. Addig együtt kell élni ezzel (és még számtalan más) korlátozással.
Néhány gondolat a további sort-paraméterekről. Az order={“ascending” vagy ”descending”} attribútummal állíthatjuk be a kimeneti sorrendet. Nekünk magyaroknak érdekes lehet még a sort lang attribútuma. Az alábbi töredéket rendeztetve

```
   <edesseg>Csoki</edesseg>
   <edesseg>Cukor</edesseg>
```

ezt kapjuk (legalábbis angol system default locale mellett):

```
Csoki, Cukor,
```

ami nyilvánvalóan nem helyes a magyar helyesírás szabályai szerint, hisz a „cs” a „c” után jönne. Ebben segít a lang attribútum:

```
<xsl:sort select="." lang="hu"/>
```

Így a rendezés már magyar logikával fog működni.

Egy kis tesztelési útmutató

Az eddigi példákat böngészőben próbáltuk ki, de bonyolultabb XSLT-knél nem mindig kapunk egyértelmű jelzéseket hibák esetén, ezért érdemes megtanulni az MSXSL használatát [7]. Ez egy konzolalapú xslt-tesztelő alkalmazás. Használata roppant egyszerű:

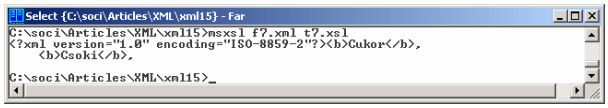
```
Msxsl.exe forrás.xml transzformáció.xsl
```

(Az msxsl nem értelmezi a forrásdoksi <?xml-stylesheet …> vezérlő utasítását.)
Az előbbi példánk kimenete így néz ki msxsl-lel:

Mik azok a furcsa üres helyek a karakterek között? A válasz az első sor encoding attribútumában keresendő. Alapértelmezett módon a transzformáció kimenete UTF-16 kódolású, azaz egyfajta unicode formátumú. Ez sokszor nem kényelmes, valamilyen karakterenként egy bájtot használó kódolásra lenne szükségünk. Ezt az output xslt elemben adhatjuk meg:

```
<xsl:transform version='1.0'... >
  <xsl:output encoding="ISO-8859-2"/>
```

Így a kimenet már a megadott kódolású lesz:



Természetesen a kimenetet gyakran átirányítjuk fájlba, így könnyebb kielemezni, azt kaptuk-e, amit terveztünk.

Fejlettebb XSLT

Az alapokat már ismerjük, itt az ideje néhány fejlettebb technikát is tanulmányoznunk.
Láttuk, hogy mind for-each, mind template-ek segítségével fel tudunk dolgozni node setet, így adódik a kérdés, melyiket használjuk? Aki csak procedurálisan hajlandó gondolkodni, az természetesen a for-each-et fogja választani. Azonban (általában) a for-each sokkal érzékenyebb a transzformálandó dokumentum szerkezeti változásaira, mind a template-es megoldások. Például az

```
<xsl:for-each select="hello/edesseg">
```

csakis a hello elemek alatti edesseg elemeket hajlandó feldolgozni, más szinten levőket nem. Ezzel szemben template-ekkel sokkal rugalmasabb transzformációkat is írhatunk, amelyek nem annyira érzékenyek a bemeneti struktúrára.
Egyből ugorjuk bele a lehető legflexibilisebb template-példába:

```
<xsl:template match="/ | @* | node()">
  <xsl:copy>
    <xsl:apply-templates select="@* | node()"/>
  </xsl:copy>
</xsl:template>
```

Vadul néz ki, ugye? Engedjük rá erre:

```
<?xml version='1.0' encoding="ISO8859-2" ?>
<hello>
   <edesseg>Csoki</edesseg>
   <edesseg>Cukor</edesseg>
</hello>
```

Kimenet:

```
<?xml version="1.0" encoding="ISO-8859-2"?><hello>
   <edesseg>Csoki</edesseg>
   <edesseg>Cukor</edesseg>
</hello>
```

Ejha, ez nagyon hasonlít a bemenetre! Sőt, szinte megegyezik, csak a hello elem átköltözött az első sorba.
Helyhiány miatt nem mutatok be más példákat, de ez a transzformáció bármely bemenetet képes megismételni, ezért is hívják *identitiás*-transzformációnak.
Hogyan működik? Intuitíven megfogalmazva rekurzívan bejárja a forrás összes xml konstrukcióját, és egyesével átmásolja őket. Közelebbről ez úgy néz ki, hogy az első és egyetlen template-ünk képes feldolgozni a dokumentum kezdetét (/), valamint harap az összes attribútumra (@*), és az összes node-ra (node()). A node() hatókörébe nem tartoznak bele az attribútumok, ezért kellett őket külön kiírni (XSL Patternsben még nem így volt, MSXSL 2.6-ban (IE5) még nem így működött!). A | (pipe) jel a halmazok unióját jelenti.
A dokumentum kezdetén tehát elindul a template. A copy elem arra utasítja a processzort, hogy másolja át a kimenetbe az aktuális node-ot (a Current Contextet). Ráadásul a copy úgy van megalkotva, hogy amellett, hogy átmásolja az aktuális node-ot, még belemásolja a törzsében megadott tartalmat is. Azaz első körben átmásolja azt, hogy elkezdődött a dokumentum (kimegy az xml deklaráció).
De mi lesz a belsejében? Az apply-templates kiválasztja az adott szinten (azaz dokumentumszinten) az összes node-ot vagy attribútumot. Ebbe belefférnek a vezérlőutasítások, kommentek és maga a gyökérelem is. Attribútum ezen a szinten nincs, hisz az attribútumok elemekhez tartoznak, és most még nem fúrtunk le egy elemig.
Tehát az előbbi példánkra alkalmazva az apply-templates által összegyűjtött node halmazban a hello elem lesz. Ki fogja feldolgozni az apply-templates által felajánlott node-ot? Nos, egy olyan template, aki képes feldolgozni a hello nevű, elem típusú node-ot. Ki lesz az? Természetesen az egyetlen template-ünk, hisz node() egyezést mutat az elemünkkel. Ezért újra elindul a sablon. A copy átmásolja a hello-t a kimenetbe, így ezen a szinten így nézne ki a kimeneti dokumentum:

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

Szándékosan nem zártam le a hello elemet, mert lehet, hogy még vannak attribútumai. Ha vannak, a copy, a sablon és a @* együttműködésével azok is kikötnek a kimenetben, a hello elem belsejében.
Azt hiszem nem kell további boncolgatnunk a példát, lehet érezni, hogy az indentitás-transzformáció a forrásdoksit csavarjaira bontja, hogy aztán újra, a megfelelő sorrendben összerakhassa. A hello elől a soremelés azért tűnt el, mert az xml parser a forrás beolvasása közben eldobja a nem szignifikáns (lényegtelen) whitespace karaktereket, azaz azokat, amelyek nem hordoznak információt. A kacsacsőrös, serializált xml formátumból memóriabeli infosetet épít, amiben már nincsenek benne sem a kacsacsőrök, sem a lényegtelen whitespace-ek.

Látszólag az identitás trafó akadémiai példa, azaz semmi haszna. Azonban nagyon jó kiindulási alap olyan transzformációkhoz, amelyekben a forrásadatok nagyrészét változatlanul kell átvinni a kimenetre, csak esetleg egyes faágakat ki kell szűrni, vagy egyes elemeket át kell nevezni.
Ha például egy vásárlókat, és a hozzájuk tartozó megrendeléseket tároló xml struktúrából ki szeretnénk szűrni az *1997*-nél régebbi *megrendeléseket*, ki kell egészíteni az identitás-transzformációt az alábbi sablonnal:

```
<!-- Ezek __NEM__ lesznek benne a kimenetben -->
<xsl:template match="orders[number(substring(
 ↳OrderDate, 1, 4)) &lt; 1997]">
  <xsl:comment>Teszt komment, ez jelenik meg
  a kiszűrt node-ok helyén -->
  </xsl:comment>
</xsl:template>
```

Ez a sablon figyel a []-ben leírt feltételnek megfelelő orders elemekre. Amikor egy ilyenhez ér az identitás-transzformáció apply-templates eleme, mindkét sablon felhatalmazottnak érzi magát, hogy lekezelje az orders elemet. Ilyen ütközéseknél a konkrétabb match kifejezést alkalmazó template kapja meg a vezérlést, azaz az új sablonunk. Az meg egyszerűen lenyeli a teljes elemet, azaz nem hívja meg rekurzívan a korábbi másolósablont. Hát nem varázslatos? És ez még csak a kezdet...

A cikkben szereplő URL-ek:
[1]: Letölthető példakódok http://technet.netacademia.net/download/xml
[2]: XML Infoset
[3]: XSL szabvány http://www.w3.org/TR/xsl
[4]: XSLT szabvány http://www.w3.org/TR/xslt
[5]: XPath szabvány http://www.w3.org/TR/xpath
[6]: MSXML4
[7]: MSXSL

Soczó Zsolt MCSE, MCSD, MCDBA
Zsolt.Soczo@netacademia.NET