XML Túra (2. rész)

A cikk előző részében átvettük az XML nyelv alapjait. Most meglátjuk, hogyan is lehet saját XML szabályokat alkotni, valamit a cikk végén olvashatunk keveset arról, hogy hogyan tudjuk HTML segítségével az XML-ben tárolt adatokat elérni.

Az érvényes XML dokumentum és a sémák

Mielőtt hozzákezdenénk a *DTD* tárgyalásához, elevenítsük fel az előző cikkből, hogy mit is nevezhetünk érvényes *XML* dokumentumnak. ***Érvényes XML dokumentum az olyan dokumentum, mely megfelel a felhasználó által definiált tartalmi szabályoknak.*** Egy ilyen tartalmi szabályokból álló halmazt nevezünk ***sémának***. Az *XML* séma az *XML* dokumentum típusának leírása, a megkötéseken túl korlátozásokat tartalmaz az adott típus struktúrájára és tartalmára. Nagyon sok szabványos és szabadalmaztatott *XML* séma jelent meg bizonyos megkötések leírására. Nagyon sok eszköz áll rendelkezésre mely a sémákkal szemben validálni tudja a dokumentumokat. De a sémáknak nem csak a szabványosítás a haszna, bizonyos szerkesztők a definiált szabályok segítségével meg tudják könnyíteni a dokumentumok létrehozását. Tehát a séma egyfajta „tervrajz" az *XML* dokumentumokhoz. Vegyük például a *WML* nyelvet, mely *WAP* oldalak írására alkalmas, szabványos nyelv. Ez is egy megfelelő , a *WML* szabályait rögzítő sémával ellátott *XML* dokumentum. Ismertebb *XML* sémák a *XSD* (*XML Schema Definition*), *Relax NG* és a *DTD*. A következőkben a *DTD*-ről lesz szó mivel ez az *XML 1.0* szabvány része és ezt tekinthetjük a legelterjedtebbnek.

DTD

A *DTD* az *XML* dokumentumban a fejléc után bárhová elhelyezhető. Általános formája:

1. Lista – Egy nagyon alap DTD

```
<?xml version="1.0"
➥encoding="ISO-8859-2" ?>
<?xml-stylesheet
➥type="text/css"
href="dolgozok.css" ?>
<!- XML Túra 2. rész ?
➥Dolgozók listája ->

<!DOCTYPE dolgozók
[
    <!ELEMENT dolgozók
    ➥(#PCDATA)>
    <!- a dolgozók elem
    ➥deklarációja ->
]
>
    <dolgozók>Kovács
    ➥Elek</dolgozók>
```

```
<!DOCTYPE dokumentumelem_neve
➥DTD>
```

A dokumentumelem azon elem melynek jellemzőit, egyedeit, jelölési deklarációit és feldolgozási utasításait szeretnénk definiálni. A *DTD* itt most az előző mondatban felsoroltakat tartalmazza. Persze a legvilágosabb rálátásunk akkor lesz , ha végre látunk egy példát, mely szemlélteti az elhangzottakat *(1. lista)*. Tehát az *1. Lista* egy érvényes *XML* dokumentum, melynek egyetlen eleme a ***dolgozók***. A ***dolgozók*** elemnek csak szöveges tartalma lehet. A szöveges tartalomra a `#PCDATA` kifejezés utal (***Parsed Character Data***).

Az ELEMENT-ről bővebben

Az elemtípusok deklarálásának általános alakja:

```
<!ELEMENT elem_neve tartalom>
```

Az elem nevének megadásakor ügyeljünk a kicsi és nagy betűk használatára. A tartalom a következőket adhatja meg:

`EMPTY` – üres elem :

```
<!ELEMENT URES_ELEM EMPTY>
```

`ANY` – bármilyen szöveges, vegyes tartalom:

```
<!ELEMENT BARMILYEN ANY>
```

Előfordulhat, hogy egy elem más elemeket tartalmaz, ekkor fel kell sorolni a tartalmazott elemeket:

```
<!ELEMENT dolgozó (név,
➥osztály, kor, fizetés)>
```

majd utána definiálni kell azokat *(2. Lista)*.

Tehát a *2. Lista* ***DTD***-jének szöveges változata. A `doctype` után a dokumentum ***gyökérelemének*** neve következik, majd az egyes elemek deklarációja. A dolgozók elem elemtartalommal rendelkezik, mégpedig bármennyi (*) dolgozót tartalmazhat. Ha elhagynánk a csillagot, abban az esetben a dolgozók elem csak egyetlen dolgozót tartalmazhatna. Több ilyen, az elemek számára utaló jelet használhatunk. Nulla vagy egy darab az előtte álló elemből: ? , egy vagy

2. Lista – Elemek definiálása

```
...
<!DOCTYPE dolgozók
[
    <!ELEMENT dolgozók
    ➥(dolgozó)*>
     <!ELEMENT dolgozó (név,
     ➥osztály, kor,
     ➥fizetés)>
     <!ELEMENT név (#PCDATA)>
     <!ELEMENT osztály
     ➥(#PCDATA)>
     <!ELEMENT kor (#PCDATA)>
     <!ELEMENT fizetés
     ➥(#PCDATA)>
    <!-- a dolgozók elem
    ➥deklarációja –>
]
>
...
```

3. lista – Jellemzők definiálása

```
...
<!DOCTYPE dolgozók
...
<!ATTLIST dolgozó külföldi
➥CDATA "nem" neme CDATA
➥#REQUIRED>
...
>
...
```

több: **+** , nulla vagy több: * .
A dolgozó elem a név, osztály, kor, fizetés elemeket fogja tartalmazni, sorrendben.
A ***sorrend*** fontos. Mivel nem használ tunk semmilyen elemszámra utaló jelölést, így mindegyiket pontosan ***egyszer*** kell hogy tartalmazza. Ha most a dolgozó elem definíciójában a vesszőket | -ra cseréljük akkor a következő

```
<!ELEMENT dolgozó(név | osztály
➥| kor | fizetés )>
```

definíció azt fogja jelenteni, hogy a dolgozó elem a felsoroltak közül legalább az egyiket tartalmazza.

Jellemzők

Jellemzők definiálása a következőképpen lehetséges:

```
<!ATTLIST ELEM_NEVE
➥jellemző_neve jellemző_típusa
➥jellemző_alapértelmezett_tart
➥alma ...>
```

Az `elem neve` azon elem, melynek éppen a jellemzőit szeretnénk megszabni. A következő a sorban a `jellemző neve` majd az `adattípusa` és végül az `alapértelmezett értéke`, melyet akkor vesz fel ha nem adják meg az értékét. Az alapértelmezésnél adhatjuk meg, hogy az éppen soron lévő jellemző megadása kötelező vagy sem. Lássunk erre is példát: *(3. Lista)*.
Azaz a *3. Lista* szerint a dolgozó elem külföldi nevű jellemzőjének alapértelmezésben nem az értéke és a neme nevű jellemző megadása kötelező.
Az egyes jellemző típusait ***három*** nagy csoportba sorolhatjuk: ***karakterlánc, token típusú és felsorolás típusú***.
Nézzük őket szépen sorjában.
A ***karakterlánc*** a fentiekben a *3. lista* alapján ki lett tárgyalva. A ***token típusok*** egyike az ***ID***. Az ilyen típusú jellemző egyfajta kulcsként szolgál, tehát két jellemzőnek mely ***ID*** típusú, nem lehet azonos tartalma:

```
<!ATTLIST AUTÓ motorszám ID
➥#REQUIRED>
...
<AUTÓ motorszám="386">
<AUTÓ motorszám="486">
<AUTÓ motorszám="386"> <!--
HELYTELEN, MERT MÁR LÉTEZIK
➥ILYEN MOTORSZÁMÚ AUTÓ!!! -->
...
```

A második token típus az `IDREF`, mely tulajdonképpen egy mutató egy már létező `ID` típusú jellemzőre:

```
<!ATTLIST SZÜLŐ személyi_szám
➥ID #REQUIRED>
<!ATTLIST GYEREK személyi_szám
➥ID #REQUIRED szülője IDREF
➥#REQUIRED>
...
<SZÜLŐ személyi_szám=
➥"23456789">
<GYEREK személyi_szám=
➥"010100111"
szülője="23456789">
```

A harmadik az `IDREFS`, mely vehető az `IDREF` többszörösének. A fenti példát tovább fejlesztve:

```
<!ATTLIST GYEREK ... szülője
➥IDREFS  #REQUIRED>
...
<SZÜLŐ személyi_szám=
➥"23456789">
<SZÜLŐ személyi_szám=
➥"12345678">
<GYEREK személyi_szám=
➥"010100111" szülője="23456789
➥12345678">
```

Az egymás után megadott `ID` mutatókat egymástól ***szóközzel*** választjuk el. A negyedik tagja ennek a csoportnak az `NMTOKEN` típus. Értéke lehet ***pontok, számok, betűk, kötőjelek és aláhúzások sorozata***. Kettőspont is szerepelhet benne, kivétel az első karaktert. Az `NMTOKENS` ennek a sokszorosított változata. Egy jellemzőn belül több `NMTOKEN` típusú értéket adhatunk meg szóközökkel elválasztva.
Az utolsó kettő az `ENTITY` és az `ENTITIES`. Az utóbbi az első sokszorosítottja a fentebb leírt módon.
Az `ENTITY` értéke egy a *DTD*-ben már deklarált, de még nem értelmezett ***egyed*** lehet.
A token típus után van még egy típus fajta amiről nem esett szó, ez a ***felsorolás***:

```
...
<!ATTLIST dolgozó neme =(férfi
➥| nő) #REQUIRED>
...
```

A fenti példa leírja, hogy a dolgozó elem kötelező ***neme*** jellemzője vagy férfi vagy nő értéket vehet fel.
A `#REQUIRED` kulcsszó párjaként említhetjük még a `#IMPLIED`-et, mely jelzi az értelmezőnek, hogy a jellemzőt ***nem kötelező*** megadni, azaz ha elhagyják, akkor is érvényes lesz a dokumentum.

Kívül vagy belül?

Akik huzamosabb ideje foglalkoznak *SGML* nyelvekkel, azok szerintem már előre ráéreztek annak lehetőségére, hogy a *DTD*-t az azt felhasználó dokumentumon kívül is el lehessen tárolni, majd arra hivatkozni, mint például a *CSS* esetében is tettük

az előző cikkben. Egy másik állományban tárolt *DTD*-t nevezünk külső *DTD*-nek a felhasználó *XML* szempontjából. Belső *DTD*-ről beszélünk, akkor ha a definíciók abban az XML-ben vannak megírva, melyekben felhasználjuk őket, mint például az eddigi esetekben.
Tiszta megértéshez mindig jól jön egy példa *(4. lista)*.
A ***dolgozok.xml***-ben a következőképpen hivatkozhatunk a külső DTD-re:

```
...
<!DOCTYPE dolgozók SYSTEM
➥"dolgozok.dtd">
...
```

Lehetőség van a külső DTD hivatkozásával egyidejűleg még további definíciók megadására is:

```
<!DOCTYPE dolgozók SYSTEM
➥"dolgozok.dtd"
[
    <!ATTLIST név főnök (igen
    ➥| nem) #REQUIRED>
]
>
```

Ekkor a két definícióhalmaz , a külső és belső DTD unióját tekintjük a teljes dokumentum DTD-jének.

Egyed

Az *ENTITY* típus leírásánál került először szóba ez a kifejezés. Az egyedeket legkönnyebben ***konstansokként*** definiálhatjuk. Egy egyedhez társíthatunk ***DTD-t, jellemzőt, elemet***, és segítségével pedig akárhányszor ***hivatkozhatunk*** rá a dokumentumban. Ezeket szintén a *DTD*-ben kell deklarálni. Ügyes használatukkal a dokumentumunk méretét jelentősen csökkenteni tudjuk.
Itt is lehetőség adódik arra, hogy külső állományból hívjuk meg az egyed értékét. Legyen az ***egyed.xml*** tartalma:

```
<név>Szabó Ubul</név>
```

Ekkor az *5. Listában* az összetett egyed deklarációját a következőre lehet módosítani:

```
<!ENTITY összetett SYSTEM
➥"egyed.xml">
```

Fent említettük, hogy az egyed tartalmazhat *DTD* deklarációt is. Itt is ugyanúgy meghívhatunk külső *DTD*-t egy egyed értékeként. Viszont a szintaxis egy kicsit másképp fog alakulni:

```
<!ENTITY % egyed_neve
➥"<!ELEMENT ...>...">
```

A *DTD*-ben pedig a következőképpen tudunk rá hivatkozni:

```
%egyed_neve;
```

Ha külső *DTD*-t szeretnénk egy egyedre ráhúzni, akkor pedig a következő a módszer:

```
<!ENTITY % kulso_dtd SYSTEM
➥"szabalyok.dtd">
```

A DSO-ról röviden

A következőkben egy olyan ***adatkapcsolásos*** technikáról lesz szó, melyet sajnos – tudomásom szerint – a ***Linux*** alatti böngészők nagy része nem támogat, de egy másik bizonyos operációs rendszer böngészője igen. Mivel itt pedig az *XML*-ről van szó és szerintem ez a technika nagyon jól kihasználja az *XML*-t, gondoltam megér pár sort. Tehát lássuk hogyan tudunk *HTML* dokumentumba *XML* adatokat illeszteni. Nem meglepő módon kétféleképpen is megtehetjük:

```
<HTML>
...
<XML ID="kozvetlen_xml">
<!-- ide jön az XML kód -->
</XML>
...
<XML ID="kozvetett_xml"
➥SRC="forras.xml">
...
</HTML>
```

Az ilyen objektumot nevezzük *Data Source Object*-nek. Az adatainkhoz az `ID`-n keresztül fogjuk megtalálni az utat, ezért mondható, hogy ez a jellemző kötelező.
A *HTML*-ben több adatbefogadó elem is létezik: `SPAN`, `DIV`, `TABLE`. Ezek közül az első kettő csak egy elem tulajdonságait tudja megjeleníteni, például:

4. Lista – Külső DTD megvalósítása (a dolgozok.dtd tartalma)

```
<?xml version="1.0"
➥encoding="ISO-8859-2">
    <!ELEMENT dolgozók
    ➥(#PCDATA)*>
    <!ELEMENT dolgozó (név,
    ➥osztály, kor, fizetés)>
    <!ELEMENT név (#PCDATA)>
    <!ELEMENT osztály
    ➥(#PCDATA)>
    <!ELEMENT kor (#PCDATA)>
    <!ELEMENT fizetés
    ➥(#PCDATA)>
    <!ATTLIST dolgozó
    ➥külföldi CDATA "nem"
    ➥neme CDATA #REQUIRED>
```

5. lista – Egy egyszerű egyed deklarációja

```
<!DOCTYPE dolgozók
[
   ...
   <!ENTITY név "Kovács Elek">
   <!ENTITY összetett
   ➥"<név>Szabó Ubul</név>"
   ...
]
>

<dolgozók>

<dolgozó>&összetett;</dolgozó>

    <dolgozó>
     <név>&név;</név>
   </dolgozó>
</dolgozók>
```

```
...
<XML ID="dolgozokdb"
➥SRC="dolgozok.xml" >

<SPAN DATASRC="#dolgozokdb"
➥DATAFLD="név"></SPAN>
<SPAN DATASRC="#dolgozokdb"
➥DATAFLD="kor"></SPAN>
...
```

Láthatjuk, hogy a DATASRC jellemző azonosítja a ***DSO objektumot***, amíg a DATAFLD a megjelenítendő adat ***oszlopát*** adja meg. Mivel ez csak egy adat megjelenítésére szolgál, lássuk az ebből a szempontból barátságosabb megjelenítést a TABLE segítségével:

```
...
<XML ID="dolgozokdb"
➥SRC="dolgozok.xml">

<TABLE DATASRC="#dolgozokdb"
➥BORDER="1">
        <TR><TD><DIV
        ➥DATAFLD="név"></DIV>
        ➥</TD></TR>
        ...
        <TR><TD><DIV
        ➥DATAFLD="fizetés">
        ➥</DIV></TD></TR>
</TABLE>
```

Ezzel a módszerrel már minden ***dolgozók*** elembe tartozó ***dolgozó*** elem adait láthatjuk szépen táblázatba szedve.
Betekintést nyerhettünk az *XML* alapjaiba. Szerintem már sokan látják, hogy nem nehéz dolog.
A *W3C* emberei ügyesen dolgoztak, hiszen könnyen tanulható, ennek köszönhetően gyorsan elterjedt. Biztos, hogy sokan hallottak már az *RSS*-ről is (***Rich Site Summary***), mely szintén *XML* alapú, webtartalom elosztására és publikálására alkalmas formátum csoport. Főleg az újabb oldalak, hírportálok, blogok használják előszeretettel az ***Atom*** nevű *RSS* formátumot. Az *XML* nem kizárólag az interneten használt formátum. Mobiltelefonok előszeretettel alkalmazzák akár ***képekről***, akár ***videókról*** legyen szó.
Mint már említettem, az *OpenOffice* is *XML* formátumban menti a dokumentumait, majd azokat tömöríti egy állománnyá. Pár nappal ezelőtt a barátnőm fényképezőgépéről szedtem le az általa készített képeket és szemembe ötlött egy *XML* állomány, melyben a fényképezőgép tárol egy csomó információt a gép használatáról, például hány kép készült eddig, hányszor volt használva a zoom funkció, milyen a gép szoftverének verziója.
Az *MP3* lejátszók többsége is a beállításait *XML* dokumentumokban őrzi a bennük lévő memórián.
Az *XML* nagyon széleskörű eszköz. A lényeg, hogy az adatok mind ember mind számítógép által aránylag könnyen feldolgozható formában jelennek meg. Az a tulajdonsága pedig, hogy saját szabályokat, sémákat alkothatunk például *DTD*-k segítségével, csak további előnyökkel jár.
Remélem hasznos betekintést nyújthattam az olvasónak az *XML* változatos, színes világába.

Radics Péter
(peter.radics@gmail.com)

Az ELTE-n tanulok programtervező matematikus szakon. Hobbim a kosárlabda, autóvezetés, webdesign, programozás. Főleg webes alkalmazások fejlesztése érdekel. 4 éve megrögzött Linux felhasználó vagyok.