Alkalmazzunk XML-t!

Manapság rengeteget lehet hallani az XML-ről. Egyes programok reklámját olvasva néha már-már forradalmi újdonságnak tűnhet. De mi is valójában, és mire használható?

Anélkül, hogy részletekbe bocsátkoznánk: az XML a HTML-hez hasonlóan az SGML-re épülő leíró nyelv. Amiben leginkább eltér tőle, az az, hogy nincsenek egy meghatározott feladatra (mint például a HTML-nél weboldalak leírására) szolgáló tagok előre meghatározva, hanem tetszőlegesen bővíthető, lényegében csak a formai (syntactic) szabályok adottak. Ebből eredően rendkívül rugalmas, mindenre ráhúzható, és vélhetően ennek köszönhető nagy népszerűsége is.
Ezzel együtt ha valaki belenéz egy-egy XML-t tartalmazó fájlba, észreveheti, hogy a szigorú szabályok miatt rendkívül redundáns, „bőbeszédű". Sok esetben az adattartalom szinte elvész a formai elemek között. Felmerül a kérdés, hogy akkor miért használják olyan sokszor, mik az előnyei?

- Jól körülírt, szabványjellegű leírással bír, ha valaki tartja magát ehhez, akkor az általa készített XML-fájlt később és más programmal is el lehet olvasni.
- Mivel a HTML-hez hasonlóan nem bináris kódokat tartalmaz, szükség esetén ember által is olvasható, illetve írható.
- Nagyon jó elemzőkönyvtárak készültek hozzá, amiket mindenféle programozási nyelvekhez illesztettek, így viszonylag könnyen és gyorsan lehet olyan programot írni, ami egy XML-állományt be tud olvasni, emellett az előállítása is egyszerű.
- Mivel a HTML (némi megszorításokkal) az XML részhalmazának tekinthető, lehetséges a kettő kombinálása (ahogy ez a korszerű böngészőkben részben meg is történik).
- A beépített feldolgozási utasítások (process instructions, PIs) és névterek (name spaces) segítségével, valamint a külső stíluslapok felhasználásával szinte programnyelv-szerű, egészen bonyolult felhasználása is lehetséges.
- Tetszőlegesen bővíthető új elemekkel (tagokkal).

Néhány XML-alkalmazási területet is megemlítenék:

- OpenOffice adatcsere-formátumként (elég nagyok a fájlok) alkalmazzák.
- A .NET és a .GNU rendszerek szintén adatátvitelre használják.
- A fentiek régebbi és egyszerűbb változata az XMLRPC távoli függvényhívásoknál a függvények változóinak és az eredmény átvitelére is alkalmas.
- Számos program a beállítások tárolására használja.
- Nos, igen, az én HTML/PHP-előállító programom bemeneteként is ezt választottam.

Az utóbbi lesz írásom témája.

A program

Néhány évvel ezelőtt készítenem kellett néhány honlapot, de gyorsan belefáradtam, hogy állandóan ugyanazokat a HTML-tagokat gépeljem be (a vizuális szerkesztők nem tetszettek, mivel sok tekintetben korlátoznak), ezért először néhány osztályt elkészítettem PHP-ben – ezek már sok mindenben segítettek. Hamarosan azonban néhány nehézség ütötte fel a fejét: a PHP-ben nehéz igazi osztályokat készíteni, még nehezebb (legalábbis az akkori, 3.0-s változatban) mutatókat alkalmazni. Ráadásul nem sikerült megvalósítanom benne a fő célomat: olyan elemeket létrehozni, amiket utána ugyanúgy használhatok, mintha a nyelv (HTML) részei lennének.
A fentiek miatt elhatároztam, hogy egy másik programot írok. Mivel akkoriban ódzkodtam a C-től, a Perl nyelvet választottam. Igaz, hogy lassúbb, viszont a programtesztelés gyorsabb, és szinte minden könyvtár illesztését megoldották hozzá (lásd CPAN). Fő elvárásaim a következők voltak:

- Legyen „kézzel", ember által írható bemeneti formátuma.
- HTML/PHP-kimenet és önműködő előállítás jellemezze.
- A Forth nyelv szavaihoz vagy a C függvényeihez hasonlóan az egyszer már meghatározott elemek a régiekhez hasonlóan, teljes értékűen, intuitív módon használhatóak legyenek. Ezt hiányoltam a leginkább a legtöbb HTML-előállító programból.
- A HTML-tagok minden meghatározás nélkül is működjenek (azaz egy HTML-fájl apró módosításokkal, átlátszóan kerüljön át rajta. Minimális módosítása az XML szigorúbb alaki szabályai miatt szükséges, de erről később bővebben szólok).

Célkitűzéseim részben meg is valósultak. Később más feladataim adódtak, és nem foglalkoztam a HTML-lel és a weboldalkészítéssel, így egy kissé elhanyagoltam a témát. Most közzéteszem, hátha valakinek ötletet ad, esetleg kedve támad továbbfejleszteni.
Röviden a program igényeiről: én a Perl 5.05-ös változattal írtam. Azóta vannak újabbak, elvileg futnia kell velük. Kisebb nehézségeket esetleg okozhat, hogy az újabb Perl-változatok mindenáron Unicode-ban szeretnének dolgozni. Szükség esetén a kimeneten alkalmazható egy szűrő, hogy a kívánt kódkészletben kapjuk meg az eredményt (például iso-latin-2). Az `XML::Parser` modulra is szükség lesz, az általam írt `XML::Element` modulhoz. (Figyelem, később jöttem rá, hogy ilyen nevű modul valamelyik CPAN-os csomagban már létezik. Ha azt telepítettük, akkor az enyémet nevezzük vagy helyezzük át, így kerülve el az ütközést.) Amennyiben adatbázis-kötést is használni kívánunk, akkor a PostgreSQL `Pg` modulját is szükségünk lesz. Ha képátalakításokra (conversion) is szeretnénk használni, az `Image::Magick` modul is kell.
Ha ezek megvannak, akkor a programot (xtend) másoljuk valahová a burkunk elérési útjába (központi telepítésnél, több felhasználó számára */usr/bin* vagy */usr/local/bin*)), az előre létrehozott elemeimet pedig az xtend program `$global` változójában megadott könyvtárba (alapban */usr/local/httpd/xml_mod*, de ez megváltoztatható). Az ***Element.pm*** modult a ***Parser.pm***-mel azonos könyvtárban helyezzük el.

Példa az index.xml-re

```
<!DOCTYPE GENERAL [
<!ENTITY atl "gr/transp.gif">
]>
<!--  `rnyékot vető lap. Egypontos konténer.  rtékek:
      arnyek                : az árnyék szélessége
      lapszin               : lap színe
      arnyekszin            : árnyék színe
      width                 : teljes szélesség
      lalign                : lap align
      align, valign, height  : TD attribútumok a tartalomra vonatkoznak
      -->
<_def arnyek="6" lapszin="white" lalign='center' arnyekszin="gray" width="100%"
                ➥align="center" valign="center" height="">
<table align='!lalign' width="!width" cellspacing="0 " cellpadding="0 " border="0 ">
<tr><td width="!arnyek" height="!arnyek" bgcolor="!lapszin">
<img src="&atl;" width="!arnyek" height="!arnyek"/></td>
<td width="100%" height="!arnyek" bgcolor="!lapszin">
<img src="&atl;" height="!arnyek"/></td>
<td width="!arnyek" height="!arnyek">
<img src="&atl;" width="!arnyek" height="!arnyek"/></td></tr>
<tr><td width="!arnyek" bgcolor="!lapszin">
<img src="&atl;"/></td>
<td width="100%" bgcolor="!lapszin" align="!align" valign="!valign" height="!height">
<__>Ide jön a body...</__>
</td>
<td width="!arnyek" bgcolor="!arnyekszin">
<img src="&atl;"/></td></tr>
<tr><td width="!arnyek" height="!arnyek">
<img src="&atl;" width="!arnyek" height="!arnyek"/></td>
<td width="100%" height="!arnyek" bgcolor="!arnyekszin">
<img src="&atl;" height="!arnyek"/></td>
<td width="!arnyek" height="!arnyek" bgcolor="!arnyekszin">
<img src="&atl;" width="!arnyek" height="!arnyek"/></td></tr>
</table>
</_def>
```

Röviden a működésről

A program bemenetként XML formátumú fájlt vár. Mint később látni fogjuk, a működés egy XML ***MakeFile*** használata segítségével jelentős mértékben bővíthető. Célszerű is így használni, mivel néhány képesség másként nem érhető el (szándékosan írom így, nagy betűvel kezdve a szavakat a névben, hogy megkülönböztessem az „igazi” `makefile`-tól). Mivel az XML elég érzékeny a formai szabályok betartására (valójában a legkisebb hiba esetén elszáll), felhívnám a figyelmet az XML néhány sajátosságára:

- A tagoknak nyitó- és záróféllel is rendelkezniük kell. Például:
  `<valami> ... </valami>`
- Tartalom nélküli tagokat is lehet használni, de akkor önmagában kell zártnak lennie. Például a HTML img tagját így kell használni:
  `<img src='kep.jpg' />`
- A tagok minden tulajdonságát (attributum) idézőjel vagy egyszeres idézőjel (aposztróf) között kell megadni, a HTML-lel ellentétben akkor is, ha egy szóból áll.
- A teljes fájlnak is egy nyitó, illetve záró tag között kell lennie, ha más nincs, erre jó a `<html></html>`
- A formai szabályok leírásánál használt karakterek nem szerepelhetnek a szöveges törzsekben, mert ettől az elemző kiakad. Ilyenkor használjuk a `<!CDATA[[` és `]]>` határolókat, az ezek közötti részt nem elemzi.
- Megjegyzést a HTML-ben megszokott `<!--` és `-->` határolók közé írhatunk.
- Készíthetünk egységeket (entity), amiket az `&nev;` formában utóbb a tagok tulajdonságaiban és a szöveges törzsben használhatunk. A meghatározás formája:
  `<!DOCTYPE GENERAL [ <!ENTITY nev "meghatározás"> ]>`
  Ennek a fájl elején kell lennie, az első valódi tag előtt. Természetesen több egységet is alkothatunk, ekkor több `ENTITY` következik egymás után.

Az egyszer már meghatározott elemek újrafelhasználhatóságát olyan módon oldottam meg, hogy minden meghatározást külön fájlba helyeztem, ezek a tag nevét viselik, ***.xml*** kiterjesztéssel. Annak érdekében, hogy különböző szintű meghatározásaink lehessenek, a program három rétegű keresést alkalmaz: először az adott ***.xml***-fájlt keresi abban a könyvtárban, amelyből futtatjuk, azaz ahol a fő fájl helyezkedik el;

utána az úgynevezett ***site*** könyvtárban, majd a globális könyvtárban (lásd fent is). Ha megtalálja, akkor önhívó (recursive) módon kezdi el elemezni őket. Ha nem találja meg, akkor az adott tagot egy az egyben átmásolja a kimenetre: így megoldott, hogy az eredeti HTML-elemek általunk írt meghatározás nélkül is használhatóak legyenek.
Nos, mivel ez eddig elég száraz volt, lássunk egy eleven példát! Tegyük fel, hogy kialakítunk egy (elsőre egyszerű) HTML-oldalt, amit több lap alapjául szeretnénk használni. Ennek meghatározása az ***alap.xml*** fájlban a következő legyen:

```
<_def tag='alap' szin='white'>
<html>
<body bgcolor='!szin'>
 <__/>
</body>
</html>
</_def>
```

Itt két különleges tagot láthatunk: az `_def` és a `___` nevűt. Általában a „rendszer” tagokat aláhúzásjellel kezdem (ez más programnyelvekben is szokásos), a megkülönböztetés érdekében. Az `_def` amellett, hogy biztosítja az XML-nek szükséges bennfoglaló tagot, alapértelmezett tulajdonságokat tud felvenni. Ebben a példában ilyen a `szin`, amelynek értéke `white`. A másik tag az `___` nevű. Ennek az a feladata, hogy a tároló típusú tag meghatározásában kijelölje azt a helyet, ahol a felhasználáskor megadott tartalom bemásolásra kerül. Lássuk, hogyan használjuk ezt, és mindjárt érthetőbb lesz! Az ***index.xml*** fájlba írjuk be a következőket:

```
<alap>
Ide jön a tartalom.
</alap>
```

Ezután a következőképpen „fordítsuk le”:

```
xtend index.xml > index.html
```

Ekkor a következő tartalmú ***index.html*** fájlt kapjuk:

```
<html>
<body bgcolor='white'>
Ide jön a tartalom.
</body>
</html>
```

Mint látható, a meghatározásnak megfelelően kibővítette a bemenetet (ezért neveztem xtend-nek, azaz extend-nek a programot). Mivel `szin` tulajdonságot nem adtunk meg az alaptagnak, az alapértelmezettet használta. Ha most ezt írnánk:

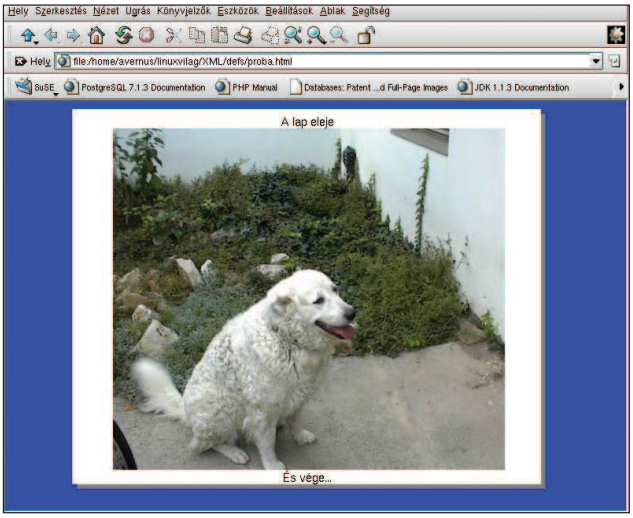
```
<alap szin='yellow'>
```

akkor a `bgcolor` értéke is ez lenne, azaz az alapértelmezettet felülírhatjuk.
Mielőtt valaki kijelentené, hogy ez eléggé fejletlen dolog, nézzünk meg az első részben egy bonyolultabb példát is. Kidolgoztam egy olyan keretet, ami úgy néz ki, mintha (megfelelő színválasztás esetén) egy papírlapon lenne a szöveg vagy más tartalom, és árnyékot vetne a háttérre. A neve legyen: lap1. Meghatározása a rendszermeghatározások között található meg, lásd *listánkon* (előző oldal).

A rendszermeghatározások közötti különbséghez képest annyit módosítottam, hogy fent az egység meghatározásában elvettem a bevezető / (per)jelet. Ez akkor szükséges, ha az oldalt HTTP-kiszolgálón keresztül kérjük le, és a képeket a törzskönyvtárból induló ***gr*** könyvtárban helyezzük el, nem pedig az adott fájl könyvtára alatt.
Ezután a következő bemeneti fájl segítségével próbáljuk is ki:

```
<html>
<body bgcolor='blue'>
<lap1 width='80%'>
A lap eleje <br/>
<img src='gr/kutyus.jpg' />
<br/>
 s vége...
</lap1>
</body>
</html>
```

Ezt lefordítva a *képünkön* látható eredményhez jutunk.



Mint látható, egy meglehetősen összetett HTML-fájlt kaptunk, a bemenet viszont kifejezetten egyszerű. Természetesen azt is megtehetjük, hogy a lap leírásából még több részt viszünk át az alapmeghatározásba: ebben az esetben az is lehetséges, hogy az egyedi fájlokban most csak a `<lap1>` és `</lap1>` közötti részeket írjuk be.
Sorozatunk következő részében megnézzük, hogyan tudjuk az XML ***MakeFile*** segítségével egy teljes honlap oldalszerkezetét leírni, milyen módon segít ez a különböző navigációs menük elkészítésében, valamint további elemekkel ismerkedünk meg. A későbbiek folyamán sorra kerül a program mint képkezelő (átméretezés, vágás, montázskészítés stb.) alkalmazása, valamint az SQL-illesztés, amivel nagyon elegánsan készíthetünk például árlistákat. Észrevételeket a hf@hmvhely.hu címre várok.

Havránek Ferenc
Automatikamérnökként dolgozik. Kedvtelései közé tartozik mindenféle kétkerekű járművön (kerékpár és motor) való közlekedés. Ezenkívül szívesen tölti idejét programozással, nem csak PC-s, hanem egyéb környezetben is, például mikrovezérlő programokat ír.