Alkalmazzunk XML-t! (2. rész)

Sorozatunk első részében megismertük az xtend programot, valamint XML nyelvű bemeneti fájljait. A mostani részben további elemeket láthatunk, bepillantunk az XML belső működésébe, emellett az XML MakeFile használatát is kipróbáljuk.

Mint korábban láthattuk, a tagok meghatározásánál egy-egy tulajdonság értékére a '!név' forma használatával hivatkozhatunk. Erre létezik egy másik lehetőség is, ekkor a '?név' alakot használjuk. Ebben az esetben az adott tulajdonságot csak akkor írja be a kimeneti tagba, ha annak az értéke nem üres. Ilyen módon, ha a meghatározásban üres értéket adunk meg neki, akkor a kimenetben csak akkor jelenik meg, ha meghíváskor valódi értéket kapott.
Mielőtt megismernénk néhány további különleges tagot, megemlíteném, hogy mindig is nagy hatást gyakorolt rám a Forth nyelvnek az a sajátsága, hogy az alapvető (mag) részeken kívül a többi részt a legtöbb esetben már magában a Forthban írják. Ezért én is arra törekedtem, hogy a programom ilyen módon kívülről, XML-tagok révén bővíthető legyen. Ezért kézenfekvő volt, hogy egy olyan tagot készítsek (a neve <_code>), ami a tartalmát végrehajtatja a Perllel (ebből fakadóan ezt a programot nem célszerű mindenki számára futtathatóvá tenni az interneten át). Így anélkül, hogy magába a programba írnánk bele, jelentős mértékben bővíthetjük a képességeit. A továbbiakban ismertetendő különleges tagok jelentős része már így is készült. Nem hallgatom el azért, hogy néhányszor mégis bele kellett írnom magába a programba, de ez csak a megvalósítás tökéletlenségét mutatja, az elvét nem.
Példaként nézzük a legegyszerűbb tagot! Ennek mindössze az a célja, hogy ha nem akarunk mást, legyen egy bennfoglaló tagunk, amit az XML minden fájl esetén megkövetel. Ez a _dummy megtalálható az ***_dummy.xml*** fájlban.

```
<_code>
$r.=xpchild($stack[0]);
</_code>
```

Rövid magyarázat is elkél hozzá: a $r globális változóban gyűjtöm a kimenetet. A különböző tagok egymásba ágyazását úgy dolgozza fel, hogy a pillanatnyi tagot egy veremtárba helyezi el, és utána tér át annak „gyermek" (bennfoglalt) tagjaira. A veremtárváltozó neve stack, és ennek a legfelső (nulladik) eleme az éppen időszerű tag – a fenti sor ennek a bennfoglalt részét bontja ki (xpchild).
Hasonlóképpen az is lehetséges, hogy egy tag tulajdonságát szintén programkód segítségével állítsuk elő; ekkor a tulajdonságot az @ jellel kell kezdeni, és a kódnak egy szöveget kell eredményül adnia, ez lesz a tulajdonság értéke (például sprintf függvény).
Ha az <_code> tagon belül olyan karaktereket használunk, amelyek megzavarhatják az XML-elemző működését, akkor a kódot tegyük az előző részben már említett CDATA határolók közé, azaz

```
<![CDATA[     kód      ]]>
```

Ejtsünk még szót néhány olyan elemről, amelyek a programozási nyelvekhez hasonlóvá teszik az XML-bemenetet. Ezekkel a kimenet előállítása közben ciklust lehet szervezni, illetve feltételes kimenetet lehet előállítani. Természetesen igazi értelmüket akkor nyerik el, ha a következő részben ismertetendő adatbázis-, illetve grafikai illesztéssel együtt alkalmazzuk őket. Mindenesetre álljanak itt, alkalmazásukra utaló környezetben:

```
<_if test='$a<3'>
  ...valami, igaz ág...
</_if>
<_else>
  .... másvalami, hamis ág...
</_else>
```

A <_else> elemnek közvetlenül az <_if> elem után kell következnie!

```
<_while test='$a<5'>
 ... kimenetet előállító rész ...
<_code>$a+=1; </_code>
</_while>
```

Hasonló módon működik az <_until> is. Ezeken kívül a switch/case szerkezet is létezik:

```
<_switch test='$a'>
<_case test='1'> ..... </_case>
<_case test='valami'> ....  </_case>
<_default> ....  </_default>
</_switch>
```

Itt kell megemlítenem az <_ins> tagot is – ez teszi lehetővé, hogy az olyan részeket, amiket nem akarunk külön előállítani, hanem már készen állnak egy külső állományban, bemásolhassuk a kimenetbe. Ilyen lehet például a stílusleírás vagy a JavaScript, esetleg más betétek, részletek. A fájlnevet a következő módon kell megadnunk:

```
<_ins file='valami.scr'/>
```

Még egy érdekes eleme van az XML-nek, amit a későbbiek folyamán több esetben is alkalmazni fogunk. Még kellemesebbé teszi az a képessége, hogy ennek révén nemcsak a HTML-elemek, hanem a PHP-betétek, kódrészletek szempontjából is teljesen átlátszó lesz a program működése; ezeket is alkalmazhatjuk a bemenetben, és változás nélkül átkerülnek a kimenetbe (amit ebben az esetben ***.php*** kiterjesztésűre célszerű készíteni, így a kiszolgálónk megfelelően fogja értelmezni). A nyelvnek ez a tulajdonsága az úgynevezett feldol-

gozási utasítás (Process Instruction, azaz PI), formája pedig a következőképpen fest:

```
<?nev ... kód ... ?>
```

A név helyére a megfelelő PI nevét kell beírni, a kód helyére pedig bármit, amit az adott nevű PI-t feldolgozó program fogadni tud. Mint látjuk, ez (talán nem véletlenül) tökéletesen megegyezik azzal a móddal, ahogyan a PHP-kódot a HTML-oldalakba általában be tudjuk szúrni. Ebben az esetben a PI neve ***php***, ezt az `xtend` program külön kezeli: változtatás nélkül átmásolja a kimenetre.
A többi PI esetében másként járunk el. A névvel ellentétben ugyanis nem feldolgozási utasításnak használom őket, hanem arra, hogy a kimenet egy részét (a PI-kben megadottakat) bizonyos csatornákba irányítsam át. A PI-k tartalma ebben az esetben Perl-kód, amelynek a kiértékelése során állítjuk be a `$k` változót, és ennek értéke adódik hozzá a PI nevével címzett csatornához; például a következő esetben:

```
<?jsa1 $k=sprintf("parent.kep.zoom(1.2);\n";  ?>
```

A `"parent.kep.zoom(1.2);"` szöveg (plusz újsor) adódik hozzá a `jsa1` nevű csatorna tartalmához. Felmerülhet a kérdés, hogy mire is jó mindez. Nos, gondoljunk bele például abba, hogy a HTML-oldalakba sok esetben olyan részek vannak beágyazva, amelyeknek a helye meghatározott, viszont meghatározásainkban több elemnek is szüksége lehet arra, hogy ezekbe adatokat írjon. Ilyen például a stíluslap, ami vagy külön fájlban, vagy pedig a HTML-oldal fejrészében kap helyet. Ha nem lennének csatornák, akkor a HTML-oldal törzsében lévő elemek nem tudnának a fejrészbe adatot juttatni. Így viszont a következőképpen játszódik le a folyamat: a program két menetben olvassa a bemenetet: az elsőben előállításra kerülnek a csatornák tartalmai, a másodikban pedig ezeket szúrja be a megfelelő helyekre. A beszúró utasítás formája így fest:

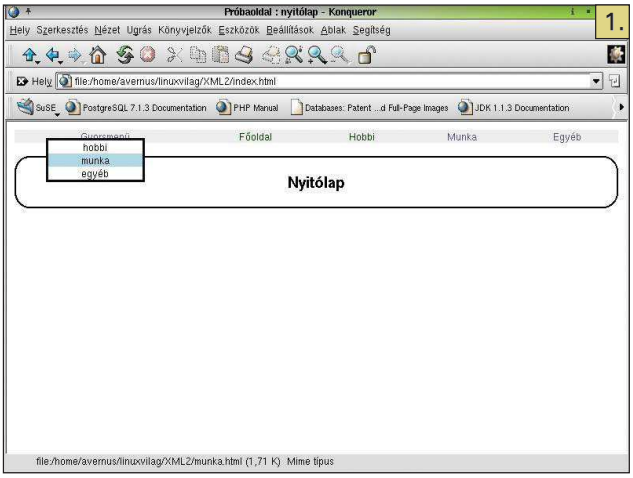
```
<_IR channel='jsa1' />
```

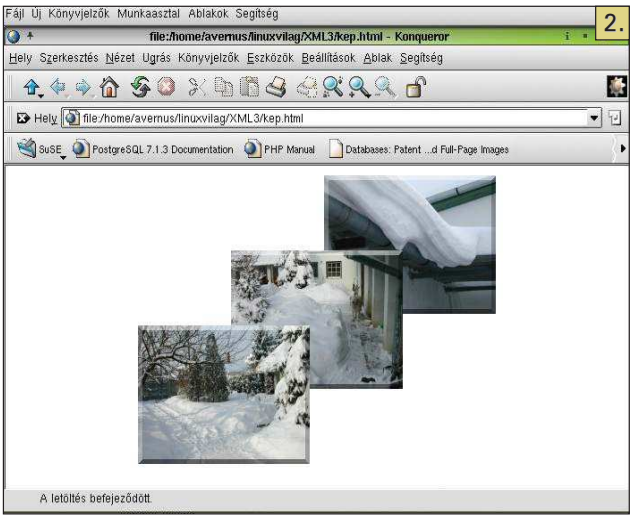
A `channel` tulajdonság értéke értelemszerűen a megfelelő csatorna neve. Még egy példa, ahol szükség van a csatornákra: a dinamikus HTML-oldalak előállításánál, ahol a JavaScript-betétek több (logikai) rétegben kerülnek elhelyezésre, és a különböző elemeknek bele kell írniuk egyikbe, másikba, többe is. Ez lineáris kimenetnél nem menne, a csatornákkal viszont egyszerű (példát a későbbiekben hozok rá).

A MakeFile.xml

A következőkben tekintsük meg a szintén xml formátumban írt `MakeFile` kialakítását. Mivel annyira magával ragadott az az elképzelés, hogy a program minden bemenő adata azonos formában kell legyen, és mivel amúgy is minden adott volt ehhez, a `MakeFile` formátuma ugyanaz lett, mint a többi bemeneté. Sőt gyakorlatilag ugyanaz a program elemzi ezt is, mégpedig néhány egyedi, e célból kialakított tag segítségével. Mi az előnye egy ilyen különleges fájl használatának azzal szemben, hogy a „valódi” `make` program „valódi” `Makefile`-jába írjuk be az előállítás feltételeit?

- Nem kell más formátumra figyelnünk (akinek nincs gyakorlata a `makefile`-ok írásában, annak ez elég komoly feladat).
- A kívánt honlap- vagy HTML-leírás oldalelrendezését is egy menetben meghatározhatjuk, ráadásul a későbbiekben ezt az adatot felhasználva a program különböző menüket és hivatkozásokat készíthet az oldalakhoz.
- Ugyanazt az egyetlen programot (`xtend`) kell meghívni az előállításkor.
- Változtatás esetén az oldal szerkezetének megfelelően újra létre lehet hozni akár egy oldalt is és az alárendelt (leszármazott) oldalakat.





Lássunk egy példát, ami némi magyarázattal kiegészítve megvilágítja a részleteket.

```
<site name='Próbalap' start='index.html'
            sitedir='./defs'>
<page name='nyitólap' src='proba.xml'
            target='index.html'>
<page name='hobbi' src='hobbi.xml'
            target='hobbi.html'/>
<page name='munka' src='munka.xml'
            target='munka.html'/>
<page name='egyéb' src='egyeb.xml'
            target='egyeb.html'/>
</page>
</site>
```

Mint látható, a `site` elem tartalmazza az összes többit. Itt adjuk meg a honlap nevét, az induló oldal fájlnevét, valamint

azt, hogy a második helyen keresett meghatározások hol találhatók (ebben az esetben a `MakeFile`-t tartalmazó könyvtárból nyíló ***defs*** alkönyvtárban).
Ezután jönnek az egyes oldalak meghatározásait leíró elemek, a `page` tagok. Figyeljük meg, hogy ezek az egymásba ágyazások révén faszerkezetszerűen írják le a teljes honlap szerkezetét. Példánkban a nyitólap a fa gyökere, és belőle nyílik a három másik oldal. Természetesen ennél sokkal mélyebben egymásba ágyazott, bonyolultabb szerkezet is előfordulhat.
Az egyes lapok esetében a három tulajdonság: az oldal neve (ezt bizonyos módszerekkel a böngésző címsorába is fel lehet vinni), a bemeneti fájl neve és a kimenet neve.
Ha ezek után a teljes honlapot elő akarjuk állítani a bemenetből, elég az alábbi parancsot kiadni:

```
xtend MakeFile.xml
```

Az `xtend` sorban előállítja a kimeneteket. Mivel eközben kiírja, hogy melyik bemeneti fájlból éppen melyik kimenetit készíti, nyomon követhetjük, hogyan járja be a fent említett faszerkezetet. Az előállítást a következő kapcsolókkal tudjuk irányítani:

`-global elérési_út` – másik keresési könyvtárat adhatunk meg.
`-site elérési_út` – a fenti `sitedir` tulajdonság helyett használható.
`-lsep 'karakter'` – a sortöréskarakter megadása, alapértelmezetten `\n`. Bizonyos esetekben, például ha olvashatatlanná akarjuk tenni a kimenetet, vagy bizonyos, erősen grafikus HTML-oldalaknál célszerű semmire ('') állítani át, így egyetlen sor lesz a kimenet, illetve a PI-k egy része ekkor is soremeléseket ír ki (ezt a JavaScript igényli).
`-x kimeneti_fájlnév` – csak a megadott és az abból leszármazott fájlokat hozza létre.
`-o kimeneti_fájlnév` – csak a megadott fájlt hozza létre (még a leszármazottakat sem).

Mivel a különböző oldalakat úgy állítja elő, hogy saját magát ismételten meghívja rájuk – azaz az `xtend` programot a megfelelő kapcsolókkal és nevekkel –, keresni kellett valamilyen megoldást arra nézvést, hogy a szükséges adatokat átjuttassuk a programhívások között (például a pillanatnyi oldal neve, a honlap neve). Ezek a hívás előtt a burok környezeti változóiban kerülnek elhelyezésre, és onnan tudja őket a program kivenni.
Ennyi szöveg után lássunk egy példát! A forrásszöveg a mellékletben található, itt csak a lényeges elemeket ragadom ki.
A nyitóoldal (***index.html***) forrása:

```
<alaplap>
<fejlec1/>
<br/>
<lap2>
<h2>Nyitólap</h2>
</lap2>
</alaplap>
```

Nagyon hasonlít hozzá a három másik oldal forrása is, csupán a `fejlec1` helyett `fejlec2` tag szerepel, és más a szöveg.
Az előállított eredmény az *1. képen* látható. A pillanatfelvétel időpontjában a gyorsmenü megnyitott állapotban található.
Ehhez a fenti kis forráson kívül másra is szükség van: egyrészről egy ***javacript*** könyvtárra, amit szintén e cikk szerzője készített, elemzése azonban meghaladja a cikk kereteit. A lényeg, hogy segítségével a 4.6-os Netscape, a 4.0-s Explorer és a 2.0-s Konquerornál frissebb, illetve az új DOM-mal (Document Object Model) egyező, JavaScriptet futtatni képes böngészőben ilyen menüket, valamint mozgó elemeket (sprite) lehet vele létrehozni. Emellett a fenti kis meghatározásban lévő egyéb elemeket is meg kell határoznunk. Példaként lássuk a `fejlec1` meghatározását (***fejlec1.xml***):

```
<_dummy>
<fwdmenu top='20' left='50' border-
color='#111111'/>
<fejfejhivatkozas>
<R><popa menu='fwdmenu'>Gyorsmenü</popa></R>
<R><a href='index.html'>Főoldal</a></R>
<R><a href='hobbi.html'>Hobbi</a></R>
<R><a href='munka.html>Munka</a></R>
<R><a href='egyeb.html'>Egyéb</a></R>
</fejhivatkozas>
</_dummy>
```

Az `fwdmenu` előre megadott elem, a `MakeFile` alapján készíti el a program, és a meghívó fájl összes leszármazott oldalát tartalmazza, ebben az esetben a fenti hármat. Ez nagymértékben a ***javascript*** könyvtárra épül. A `fejhivatkozas` egy olyan felsorolás, amelybe változó számú elemet lehet elhelyezni, és a kimeneten megismétli a meghatározásában megadott részeket. Ebben az esetben ezek táblacellák (td), teljes meghatározása a tábla leírását is tartalmazza.
Egy, az `fwdmenu`-höz hasonló feladatú elemet fedezhetünk fel a három beágyazott oldalon. Ez az úgynevezett `sitemenu`, tulajdonképpen az adott oldalhoz vezető útvonalat adja meg, hivatkozásként feltüntetve az elágazási pontokat. Ha megfelelő logikával készítettük el a `makeFile`-unkat, ezzel a kettővel megtakaríthatjuk a honlap „térképének” a különböző oldalakon történő elhelyezését, hiszen az `fwdmenu` megadja, hogy az adott oldalról hová mehetünk, a `sitemenu` pedig azt, hogy honnan jöttünk.
Hasonlóképpen a program a `MakeFile` alapján készíti el a böngésző címsorába kerülő címeket is. Ez, ha egyszer az `alaplap` elem meghatározásában a megfelelő kódrészletet elhelyeztük, önműködő.
A csatornák használatát az `alaplap` meghatározásában követhetjük nyomon. Ez írja ki a `jsb1`, `jsa1`, `jsa2`, `jsa3` és `style` csatornák tartalmát. Ezekbe a menüket előállító elemek írnak JavaScript-kódokat, valamint stílusadatokat.
Mint látjuk, egy viszonylag jól átgondolt elv szerint működő program esetén a már egyszer elkészített elemek újrafelhasználása nagymértékben leegyszerűsödik. Mégpedig nemcsak a ma szokásos kiszolgálóoldali programok módszerével, hanem statikus oldalak (például CD-re készülő katalógus) esetében is.
A tényleges megvalósításom hibákat és az elvtől való eltéréseket is tartalmaz. Ezekről és a program adatbázis-kezelő (PostgreSQL), valamint képfeldolgozó (ImageMagick) alkalmazásáról a következő részben olvashatunk.

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