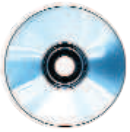


Alkalmazzunk XML-t! (3. rész)

Ebben a részben szót ejtünk a program hiányosságairól, azután a Perl SQL-csatlakozás használatáról, végül pedig készülő weblapjaink képanyagának az elkészítéséről.

© Kiskapu Kft. Minden jog fenntartva

Elsőként – nem hagyhatom a végére, nehogy a kedvezőtlen benyomás maradjon meg – a hiányosságokról kell szólnom. Programunk felépítésének az a legfőbb hibája, hogyha egy tároló típusú elemet (amilyen például az előző részekben említett `lap1`) egy másik elem meghatározásában szeretnénk használni, az csak akkor működik az elképzeléseink szerint, azaz a végső hívásból véve a kimenet tartalmát, ha mi magunk (feltehetően hosszas gondolkodás után) megadjuk, hogy milyen szintről (veremszint) vegye a kimeneti adatot. A szintet a `level` kapcsolóval adhatjuk meg. Ez a hiányosság természetesen nagyon zavaró, de mivel csupán az elemek meghatározása során bukkan fel és a végső használatnál nem, úgy gondoltam, hogy ennek ellenére a program a gyakorlatban használható.
A másik nehézség abból adódik, hogy a program Perl-parancsfájlként működik, ennek megfelelően (különösen az első indításkor) elég lassú. Ezt hivatott kiküszöbölni a `-x` és `-o` kapcsoló, amelyek korlátozzák az elkészítendő fájlok számát. Ugyancsak gond, hogy bármilyen jó is az XML `MakeFile` a honlap oldalfelépítésének a leírására, nem biztosítja azt a rugalmasságot a célfájlok készítésére, amit az „igazi” `Makefile` és a `make` program használata (például a módosítások követése). Nos, ennyi talán legyen elég az önbecsmérlésből. Ki-ki keresheti a további hibákat, remélem, használat közben, nem pedig ex cathedra…

Az SQL-illesztés

Programunkhoz, tételezzük fel, létezik egy PostgreSQL adatbázisunk, valamint telepítve vannak a megfelelő programkönyvtárak (`Pg` Perl-modul). Ezután készítsük el a következő két táblát, az alábbiak szerint:

```
create table csoport (
       kod    text,
       nev    text );

create table arlista (
       csoport      text,
       nev          text,
       ar           float );
```

Vélhetőleg egy egyszerű, cikkcsoportokra osztott árujegyzéknek megfelelő ez a két tábla. Az `arlista` táblában a `csoport` mező idegen kulcs, a csoport tábla `kod` mezőjével áll kapcsolatban, ez viszont saját kulcs. Mindenesetre használhatjuk az általam mellékelt próbaadatokat. Figyelem, az árak elavultak!
Az SQL sajátosságainak megfelelően az illesztőfelület az alábbi módon réteges felépítésű:

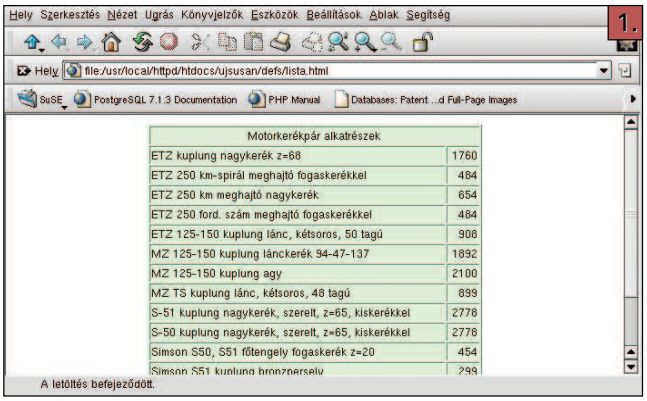
- adatbázis;
- kérdés/eredmény (query/result);
- eredménysor (row);
- mezőérték (field);

A korábban már megfigyelt rugalmasság megvalósítása érdekében itt is igyekeztem minél szabadabbá tenni a használatot. Ezt hivatottak segíteni a különböző vermek. Ezek révén lehetséges több SQL-kérdés és eredménysor keverése, a név megadásával a megfelelő kiválasztása, illetve egyszerűbb esetben az alapértelmezett alkalmazása (a verem tetején lévő). Lássunk egy egyszerű példát! Ez a fent megadott adatbázisokból készít egy táblát olyan módon, hogy cikkcsoportonként külön kiírja az árukat. Ez idő tájt láthatjuk használat közben a korábbi részekben említett `_while` tagot is.

```
<html>
<table align='center' border='1'
bgcolor='#DDFFDD'>
<_sqldb name='susandb' dbname='susan'
➥user='avernus'>
<_sqlquery name='csoportok' query='select *
➥from csoport'/>
<_sqlrow name='csoport' qname='csoportok'/>
<_while test='sqltest("is_next","csoport")'>
<tr>
➥<td colspan='2' align='center' class='cimsor'>
<_sqlfn qname='csoportok' rname='csoport'
➥f='nev'/>
</td></tr>
<_sqlquery name='aruk' query='@sprintf("select
➥* from arlista where
➥csoport=&#39;%s&#39;",sqlfn("kod","csoportok",
➥"csoport"))'/>
<_sqlrow name='aru' qname='aruk'/>
<_while test='sqltest("is_next","aru")'>
<tr><td align='left'>
<_sqlfn qname='aruk' rname='aru' f='nev'/>
</td>
<td align='right'>
<_sqlfn qname='aruk' rname='aru' f='ar'/>
</td></tr>
<_sqlrow name='aru' qname='aruk'/>
</_while>
<_sqlrow name='csoport' qname='csoportok'/>
</_while>
</_sqldb>
</table>
</html>*
```

Egy adatbázis-kapcsolatot hozunk létre az `_sqldb` taggal. A `name` érték a későbbi hivatkozásra szolgál, így tetszőleges egyedi szó lehet. A másik két érték (`dbname` és `user`) a PostgreSQL-adatbázis neve és az engedéllyel bíró felhasználónév (nincs külön azonosítás). Mint látható, az `_sqldb` tartalmazza a többi SQL-es elemet, így nem kell megadni bennük az adatbázisnevet (a verem felső eleme).
Ezután az `_sqlquery` taggal felteszünk egy kérdést az adatbázisnak (`"select * from csoport"`). Az `_sqlrow`

© Kiskapu Kft. Minden jog fenntartva



Motorkerékpár alkatrészek	
ETZ kuplung nagykerék z=68	1760
ETZ 250 km-spirál meghajtó fogaskerékkel	484
ETZ 250 km meghajtó nagykerék	654
ETZ 250 ford. szám meghajtó fogaskerékkel	484
ETZ 125-150 kuplung lánc, kétsoros, 50 tagú	908
MZ 125-150 kuplung lánckerék 94-47-137	1892
MZ 125-150 kuplung agy	2100
MZ TS kuplung lánc, kétsoros, 48 tagú	899
S-51 kuplung nagykerék, szerelt, z=65, kiskerékkel	2778
S-50 kuplung nagykerék, szerelt, z=65, kiskerékkel	2778
Simson S50, S51 főtengely fogaskerék z=20	454
Simson S51 kuplung bronzperselv	299

taggal lehívjuk az első eredménysort. Ezután az `_while` tag segítségével végigmegyünk az eredménytáblán. Ebbe a `while` ciklusba ágyazva készítünk egy másik kérdést, amely a pillanatnyi csoport tagjait adja eredményül. Ehhez sajnos használnunk kell egy beépített függvényt, amely visszaadja egy adott kérdés–eredménysor–mező hármas értékét. Mivel ez a függvény nem ismeri az XML-elemzés helyzetét, így az SQL-vermekről sincs tudomása, sajnos mindhárom értéket meg kell adni. Újabb kényelmetlenség…
Az `_sqlfn` tagokkal a kimeneten elhelyezzük a megfelelő mezők értékét (meg van adva a kérdés, a sor és a mezőnév is, mivel jelenleg nem tartalmazzák ezeket az elemeket). Ha sikerül lefordítanunk ezt az állományt, akkor az *1. képen* láthatóhoz hasonló táblázatot kapunk eredményül.
Mint látható, a csoportokat külön sorban emeli ki, a mezők megfelelően vannak igazítva. Ennek a megoldásnak az igazi szépsége abban rejlik, hogyha az adatbázis módosítása után a fájlt újrafordítjuk, akkor mindennemű pluszmunka nélkül a pillanatnyi eredményt kapjuk meg. Így könnyen el lehet – például ismertető CD írásakor – a pillanatnyi listát készíteni, és nyomban fel is lehet írni, mindezt önműködően.

A képátalakító illesztése

Létezik egy nagyszerű képátalakító programkönyvtár, mégpedig az ImageMagick. Legfőbb előnye az, hogy mindenféle programnyelvhez megoldott az illesztése, így az átalakítások remekül gépesíthetők. Én magam is használtam a C, C++ és Perl nyelvi felületét. Ráadásul az illesztőfelületek felépítése hasonló (az objektumközpontú változatok hash értéklistát tudnak fogadni). Természetesen ne várjunk olyan mérvű hatásokat, mint amilyeneket a Gimppel érhetnénk el, viszont egyszerűbb forgatásra, átméretezésre, vágásra, beillesztésre, keretezésre, képformátumváltásra kitűnően használható, viszonylag gyors és kényelmes.
E rövid bevezető után vágjunk azonnal a közepébe! A legfontosabb, gyakran használt műveleteket a szokott módon: külső XML-modulok segítségével vittem be a programba. Lássuk a modulokat:

```
<convert file='bemenetifájl_neve'
target='kimenetifájl_neve' format='formátum'
quality='szám'/>
```

Mint nyilvánvaló, ez egy nem tároló típusú elem, azaz önmagában zárt. A be- és kimenet egy-egy képfájl, a formátum a jellegzetes kiterjesztéssel kerül megadásra, például jpg vagy png. A ***quality*** tulajdonság főleg jpg-nél szükséges, a minőséget adja meg, mivel a jpg veszteséges tömörítés, és a minőség határozza meg, hogy mennyire hű az eredetihez. Természetesen a jobb minőség nagyobb fájlméretet eredményez. A 75 százalék megfelelő szokott lenni.

```
<cropper file='bemenetifájl_neve'
grdir='képek_könyvtára'> ... </cropper>
```

Ez az elem a vágástároló eleme. Itt adjuk meg a vágandó fájlt, valamint az esetleges elérési útját. Ezután az alábbi `crop` elem segítségével tudjuk darabolni. Ez a felépítés azért jó, mert sokszor egy nagyobb képet több, esetleg sok részre kell vágni a grafikus felépítésű oldalakhoz.

```
<crop target='célfájl_neve' x='xpozíció'
y='ypozíció' w='szélesség' h='magasság'/>
```

Ez a tényleges vágóelem. A fenti `cropper` elemben lehet használni. Az ott megadott fájlból vágja ki az ***xy*** pozíció és a szélesség/magasság számnégyessel megadott részletet, és a megadott célfájlba rakja. A célfájl kiterjesztése adja meg a kimeneti formátumot.

```
<placer file='bemenetifájl_neve'
target='célfájl_neve'> ... </placer>
```

Ez a beillesztés tárolóeleme. A vágással ellentétben itt már meg van adva a célfájl, így egy (változatlanul maradó) kiinduló fájlba beszúrjuk az alábbi `place` elemmel megadott részleteket, majd a `target` értékben megadott néven mentjük (a formátumot itt is a kiterjesztés adja meg).

```
<place file='forrásfájl_neve' raise='hash'
scale='hash' border='hash' rot='fok'
x='xpozíció' y='ypozíció' />
```

Íme, a beillesztőelem. Elég bonyolult, mert a beszúrás előtt műveleteket is tud végezni a bemeneti fájlokon. A `rot` érték segítségével forgathatunk, fokban lehet megadni. Megjegyzem, hogy az ImageMagick bizonyos változatainál a forgatás eltolódással is jár, ha nem 90 vagy 180 fokról van szó. Ezt ki kell próbálni. A `scale` értékkel át lehet méretezni a beillesztendő képet. A megadandó indexelő nyalábolás (hash) formája:

```
'geometry => "800x600"'
```

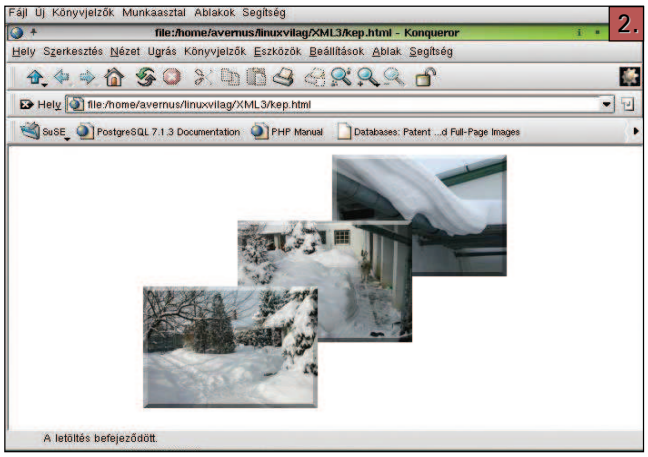
Természetesen a kívánt méreteket szükséges megadni.
A `raise` a kiemelést, illetve a besüllyesztést végzi, például

```
raise='geometry => "5x5", raise => "1"'
```

Az 1 azt jelenti, hogy kiemelésről van szó, a 0 pedig bemélyítést jelent.

```
<write x='xpozíció' y='ypozíció'
font='ttf_fontnév' size='fontméret'> ...
szöveg ... </write>
```

Ezzel az elemmel lehet a képre szöveget felírni. Ha akad TrueType betűkészletünk, azt megadva használhatjuk. A nyitó- és zárótag között kell lennie a szövegnek. Ha ez különleges karaktereket is tartalmaz, célszerű a `CDATA` határolót használni, hogy ne zavarja meg az XML-elemzőt. Sajnos jelenlegi, egyszerű formájában semmiféle tördelést nem végez a szövegen,



2.

azaz soronként kell bevinnünk, ügyelve, hogy ne lógjon le a képről (egy `write` elem/sor). Erre is lássunk egy egyszerű példát! Tegyük fel, hogy van három képünk, amelyeket lépcsőzetesen szeretnénk elhelyezni. Készítsünk a cél méretének megfelelő üres képet (például a Gimppel), legyen, mondjuk 400×300 pont. A háttért állítsuk átlátszóra. Nevezzük el ***ures.png***-nek!
Ezután használhatjuk a következő XML-fájlt a kívánt cél elérésére:

```
<_dummy>
<placer file='ures.png' target='cel.png'>
 <place file='ho1.jpg' x='208' y='0'
 scale='geometry => "192x144"'
 ➥raise='geometry => "5x5", raise => "1"'/>
 <place file='ho2.jpg' x='104' y='78'
 ➥scale='geometry => "192x144"'
 ➥raise='geometry => "5x5", raise => "1"'/>
 <place file='ho3.jpg' x='0' y='156'
 ➥scale='geometry => "192x144"'
 ➥raise='geometry => "5x5", raise => "1"'/>
</placer>
</_dummy>
```

Ha ezt az `xtend fájlnév` paranccsal lefordítjuk, akkor a *2. képet* kapjuk eredményül (ebben az esetben egy HTML-fájlba helyezve).
Azt hiszem, az alapelv érthetővé vált. Maga az xml formátum csak adathordozóként szerepelt itt. De – és ezt fontos megérteni – másutt is csak erről van szó! Az XML nem csodaszer, önmagában nem több, mint egy jól meghatározott leíró formátum. De éppen azért, mert jól meghatározott és bővíthető, és mivel kitűnő elemző könyvtárak állnak hozzá rendelkezésre, nagyon jól használható különféle programjaink adattárolására és bemeneti állományának.
További ötleteket szívesen fogadok a hf@hmvhely.hu címen.

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.

© Kiskapu Kft. Minden jog fenntartva