Az SVG világa (4. rész)
Képzeletünk határai

Most, hogy már megismertük, mire is képesek a szabad szoftverek SVG fronton, itt az ideje, hogy a horizontunkat kicsit kitágítsuk! Láttuk a fájlformátumot, a rá épülő alkalmazásokat is, de a lehetőségeknek itt korántsincs vége. Tulajdonképpen azt kezdünk az SVG-vel, amit akarunk! Számtalan programozási nyelv áll szolgálatunkra és némelyeknél már nem kell a világot újra felépítenünk magunknak.

Egyszóval elkényeztetnek minket *SVG* terén is. Persze a szabad szoftver itt is óriási plusz: a fekete doboz mentalitás helyett (vö.: bemenet-kimenet – közte a nagy homály) mindig a dolgok mélyére tudunk ásni, ami a programozás terén érdekes is és a hibakeresést is megkönnyíti. Kezdődjék az enumeratio, jöjjenek az objektumok!

A nyelvek kavalkádja és a megközelítés

SVG objektumokról szólj nékem *Google*! És szól is! Nem könnyű választani, hogy mely programnyelvekről legyen szó most, de a választás egyben szükséges és kényszerű. Szükséges, hiszen a nyelvek népszerűsége nyomán nem minden érdekli annyira a nyájas olvasót és kényszerű azért, hiszen egy Linuxvilágnyi terjedelmet is meg lehetne tölteni a különféle nyelvek *SVG*-vel kapcsolatos lehetőségeivel. Így aztán tervszerűen, előre megfontolt szándékkal a *PHP* és a *Perl* lesz a két alanya e cikknek a főszereplő *SVG* mellett. Nem magyarázkodásképpen, sokkal inkább magyarázatképpen jó tudni, hogy Magyarországon ebben az évben október 22-én először volt *Perl* konferencia, amin *Larry Wall*, a nyelv megalkotója is részt vett, valamint a *netcraft.com* szerverinformációs eszközével könnyen rájöhetünk, hogy hallatlan sok szerver van felszerelve a *PHP*-val. A *Perl* pedig hatalmas, szervezett struktúrában kínálja a nyelvhez a kiegészítő elemeket, ami nagy könnyebbség. Rutinosak a *CPAN*-ról már egyből tudják, mire is kell gondolni.

Alapvetően kétféle módon közelíthetjük meg az *SVG* formátum felhasználását programjainkban. Felhasználhatunk már meglévő *SVG* képeket, hogy azokat megjelenítsük vagy magunk állíthatunk elő *SVG* ábrákat. Az első esetnek nem sok érdekessége van, hiszen erre kész megoldások léteznek. *KDE*-re épülő alkalmazáshoz az *svg.kde.org* címen nézzünk szét, *Gnomehoz* pedig a *librsvg.sourceforge.net* címen. Sokkal érdekesebb helyzet, ha tartalmat akarunk létrehozni, ebben sokkal inkább van perspektíva!

Ma a legnépszerűbbek a szkriptnyelvek; és ahol ezeket a programokat futtatni szokás, az korántsem meglepő módon a web. Nem egészen véletlen a webre kerülő alkalmazások nagy népszerűsége, hiszen bárhol, bármikor elérhetőek és gyakorlatilag platformfüggetlenül használhatóak. A *Perl* kezdetekben uralta a webes programok felségterületét, aztán a *PHP* mára átvette ezt a stafétabotot. Gyakran hangoztatott érv, hogy a *Perl* túl lassú. De nem kell messzire menni sikeres *Perl* alapú portálokért: a népszerű *Slashdot* is ezt használja, ami kellő bizonyíték, hogy nagy terhelés mellett is lehet Perlt használni. A következőekben a *PHP*-s és *Perl*-es *SVG*-gyártásról lesz szó. A továbbiak megértéséhez az alapvető programozási fogalmak, mint változó, ciklus és függvény ismerete szükséges. Az objektumé nem feltétlen, bár azokkal fogunk dolgozni.

Az objektum, pontatlan megfogalmazásban, nem más, mint egy saját emlékezettel ellátott függvény. Az adatszerkezeteket egységbe foglalhatjuk a rajtuk műveleteket végző függvényekkel. Ez valójában egy végletesen leegyszerűsített magyarázat, de ahhoz elég, hogy az *SVG* objektumokkal dolgozni tudjunk.

A cikkben újfent alapvető dolgokról lesz szó, így az elején tekintsük át azért, hogy milyen óriási lehetőségek rejtőznek abban, ha saját programunkkal készítünk *SVG*-t! Összekapcsolhatjuk a lentieket adatbázisokkal, mobil eszközökkel(a WAP már elég elterjedt, csak éppen nincs rá meg az a húzóalkalmazás, amitől használnák is a tömegek) vagy akár a *Google API* (➲ *http://www.google.com/apis/*)-val is. Néhány ötlet az itt felsoroltakra: webáruház adatbázisának *SVG*-s vizualizációja, mobiltelefonra térkép vagy akár a *Google Fight* (➲ *www.googlefight.com*) animált *SVG*-s verziója is elkészíthető. A lehetőségeket tényleg a fantáziánk határolja be.

Perl

„Puszta kézzel" is lehetne *SVG*-ül sorokat kiírni a szabványos kimenetre, hiszen tulajdonképpen ennyiből áll a tennivaló, ezt sokkal elegánsabban

is megoldhatjuk, arról nem is beszélve, hogy ha az *SVG*-vel kellene bajlódni, az alapprobléma sikkadna el.
A már emlegetett *CPAN*-ról szóljunk egy kicsit! A *CPAN* betűszó feloldása a *Comprehensive Perl Archive Network*. Ebben a cikk írásának pillanatában 8849 modult lehetett találni. A *CPAN*-ban nem csak szoftverek, hanem dokumentációk is vannak. Ebből az óriási kavalkádból természetesen találunk *SVG*-hez kézreálló segítséget. A megtalált kiegészítést fel kell először telepíteni, amit a *CPAN* parancsértelmezővel tehetünk meg az alábbi módon:

```
perl -MCPAN -e shell;
```

A fenti parancsot root-ként adjuk ki, hiszen most programkönyvtárat fogunk telepíteni a rendszerre.
Ha először indítjuk el ebben az üzemmódban a Perlt, néhány triviális kérdésre válaszolnunk kell, hogy hogyan tudja elérni a *CPAN* archívumot, azután munkához láthatunk. A `cpan>` prompt után adjuk ki azt az `install SVG` parancsot, és ha minden rendben lezajlik, akkor minden előfeltétel adott ahhoz, hogy elkezdhessük a munkát.
Ha valakinek nincs internetkapcsolata, ez a módszer nem működőképes. Ez esetben letöltheti *tar.gz* formátumban a modult és telepítheti kézzel a *README*-ben leírtaknak megfelelően.
Ennyi bevezetés után igazán szeretnénk olyan kódot látni, ami már *SVG*-t generál. Rajzoljunk színes(véletlen alapon) köröket egy szinuszhullám mentén (*1. Lista*)!
A keletkező képet bármilyen *SVG*-képes nézegetővel/szerkesztővel meg tudjuk nyitni. A `dec2hex`, `szin`, `rgb` függvények magukért beszélnek. A `dec2hex` egy tízes számrendszerbeli számot tizenhatos számrendszerbelivé alakít, az `rgb` `00`-tól `FF`-ig ad vissza hexadecimálisban egy véletlenszámot, az rgb pedig három ilyen felhasználásával egy *SVG* értelmező számára is emészthető színkódot ad. A `dec2hex` függvényben a `? :` operátorra azért volt szükség, hiszen a 0-t `00` alakban kell megadni(és így tovább, mindig kétjegyűre bővítve), mert feltétlenül hat karakterből kell álljon az érvényes színkód.
A `$szelesseg`, `$magassag` változóban az elkészülő ábra méreteit tároljuk,

1. Lista szinusz.pl

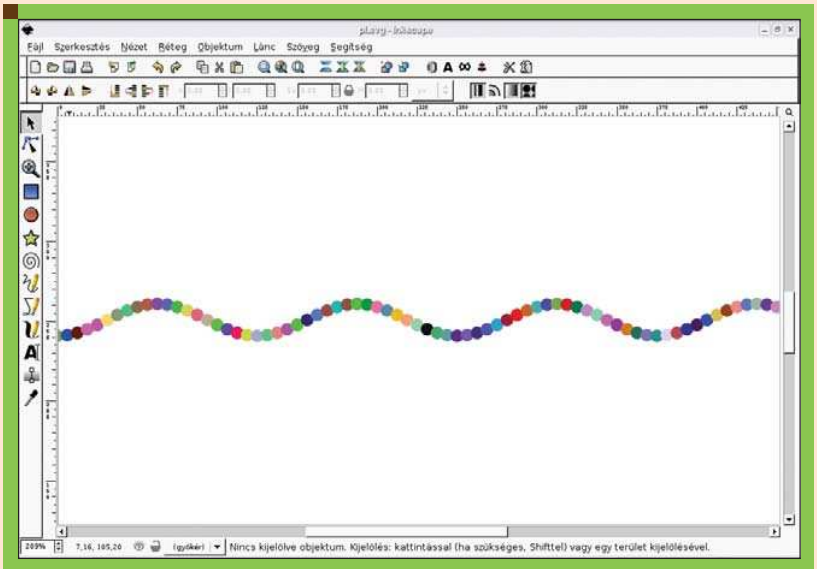
```
use SVG;

sub dec2hex($) {
   my( $dec ) = shift;
   return length(sprintf("%lx", $dec ))== 2?sprintf("%lx",
   ➥$dec ):sprintf("0%lx", $dec);
}

sub szin {
   return dec2hex(int(rand 255));
}

sub rgb {
   return "#" . szin . szin . szin;
}

$szelesseg=500;
$magassag=500;
$kordarab=80;
$PI=3.14159265;
my $svg= SVG->new('width',$szelesseg,'height',$magassag);
for($i=0; $i<$kordarab; $i++)
{
   $svg->circle(id=>'kor' . $i,
   cx=>(($szelesseg/$kordarab)*$i),
   cy=>($magassag/2)+(cos($PI*$i*0.1)*10),
   r=>$kordarab/20,
   style=>{fill=> rgb }
   );
}
print $svg->xmlify;
```



1. ábra A szinusz.pl kimenete Inkscape-ben megnyitva

a `kordarab`-ban pedig azt, hogy hány kört szeretnénk elhelyezni a szinuszhullámon.
A `for` ciklus előtt már található érdekes rész: egy új *SVG* objektumot hozunk létre, amit a `$svg`-vel jelölünk.
A létrehozáskor megadjuk a kiterjedését. A ciklusban található az egyetlen rajzoló utasítás. Itt a létrejövő új elem *XML* azonosítójára, az X és Y koordinátákra és a kitöltés színére van szükségünk. Az utolsó előtti sorban azt

1. táblázat

Függvény neve	Geometriai alakzat	Függvény paraméterei
`circle`	kör	• `cx` – középpont x koordinátája • `cy` – középpont y koordinátája • `r` – sugár
`ellipse`	ellipszis	• `cx` – középpont x koordinátája • `cy` – középpont y koordinátája • `rx` – ellipszis magassága • `ry` – ellipszis szélessége
`rectangle`	téglalap	• `x` – x koordináta • `y` – y koordináta • `width` – szélesség • `height` – magasság • `rx` – saroklekerekítés x sugara • `ry` – saroklekerekítés y sugara
`image`	bitkép beszúrása	• `x` • `y` • `width` • `height` • `'-href'` – a bitkép elérési útja

mondjuk, hogy az eddigi tényke-déseinket a `$svg` objektumot alakítsa át *SVG* formátumba és ezt írja ki a képernyőre. Ebből a kis szkript-ből is látszik, hogy sokkal több időt töltöttünk el a valódi feladattal (színkódok és szinuszhullám), mint amit az *SVG*-re fordítottunk. Ez nagyobb terjedelmű munkák esetén csak fokozódik.
Szép és jó, hogy egy kört el tudunk helyezni valahol a síkon, de az *SVG* ennél azért többet tud.
A ➲ *http://search.cpan.org/~ronan/SVG-2.33/lib/SVG/Manual.pm* oldalon találjuk a teljes referenciát, de a leg-fontosabb tudnivalók itt is olvashatók. Ahogy az *SVG*-ben is megvoltak az alapvető építőelemek, éppen úgy itt is. Az *1. táblázat* ezeket foglalja össze. Ezeket a következő formában lehet használni:

```
$svg_objektum->Függvény
➥neve(Függvény paramétere=>
➥"valami", Függvény
➥paramétere=>"valami");
```

A szöveg használata még fontos lehet az elemi alkalmazásoknál is, ez azon-ban nem sorolható be a fentiek közé formai szempontból:

```
$svg->text(x=>10, y=>10,
➥-cdata=>'LinuxVilág');
```

Persze ha a kedvenc újságunk neve helyett mást akarunk leírni, annak sincs különösebb akadálya. A betűtí-pust a most következő style használa-tával tudjuk beállítani.
Ugyan az alapokon túl vagyunk, de formázás még hiányzik. A `style` pont erre való a következőképpen:

```
style => { CSS tulajdonság=>
➥'érték', CSS tulajdonság=>
➥'érték' }
```

Ezt lehet mindegyik elemhez használ-ni, hogy mi módon, az a példaprog-ramból kiderül. Ezekből az alapvető építőelemekből és azokból amiket itt nem részleteztünk, felépíthetőek az *SVG* fájlok a *Perl* segítségével.

PHP

A *PHP*-nál nem egy hivatalos hely-ről, hanem a ➲ *www.phpclasses.net* oldalról fogjuk beszerezni az *SVG* gyártáshoz elengedhetetlen eszközt. A *PHP* alapvetően webes eszköz, így készítsünk most el egy olyan szkriptet, ami egy weboldalon be-kér egy szöveget és azt utána külön-féle színű és méretű (véletlenszeru-en) betűkkel írja fel egy SVG doku-mentumban. A szükséges fájlokat a ➲ *http://www.phpclasses.org/browse/package/457.html* címről tudjuk letölte-ni. Ez hasonlóan a Perles példához, egy objektum lesz. ami elvégzi a pisz-kos munkát *SVG* fronton. A példa ki-próbálásához nincs is másra szükség, mint egy webszerverre, ahol működik a *PHP*. Hozzunk létre egy könyvtárat például *svg* néven és oda töltsük fel a *phpclasses.org*-ról leszedett fájlokat. Ha ez megvan, az itt közölt példa-program is életre kel (*2. Lista*).
Az `SVG_CLASS_BASE`-t kell csupán úgy átírni, hogy megtalálja az objektum saját magát és ki is lehet próbálni az alkotást. A fájl elején pontosan ugyan azt kellett elvégezni, amit *Perlben* is, egy véletlenszerű hexadecimális szín-kódot előállítani. Attól függően, hogy kapott-e `POST` módon `szoveg` paramé-tert a program, vagy kiírja az űrlapot vagy előállítja az *SVG* fájlt. Az űrlap kiírása minimális *PHP* és *HTML* isme-retekkel megérthető, de az `if` előbbi ága már nem ennyire triviális.
Be kell először is állítani a már említett `SVG_CLASS_BASE`-t, hogy az objektum tudja, merre vannak a saját részei. Ezután a `require_once("Svg.php");` sorral jelezzük, hogy erre a fájlra szükségünk van a kódban. A

```
$svg =& new SvgDocument
➥($betukoz *($hossz+1) ,
➥$magassag);
```

sor létrehoz egy új példányt az objek-tumból, amivel megkezdődik a tényle-ges *SVG*-zés. A `for` ciklus által betűn-ként rakjuk a dokumentumba a szöve-get, ráadásul a színen kívül még a be-tűméretet is véletlenszerűen válasz-tjuk két határ között. A `$svg->printElement();` pedig mindent, amit a $svg-nek kiadtunk, *SVG* formá-ban visszaadja. Ezen *SVG* objektum részletes ismertetése nem célja ennek a cikknek, maga a *PHP* kódja már do-kumentálja a működését. A *Perlben* felsorolt lehetőségek mindegyike léte-zik itt is, a szintaktika tér el csak kissé. Néhány fontosabb függvény a forrás-kódból:

```
function SvgText($x=0, $y=0,
➥$text= "", $style="",
➥$transform="");
function SvgRect($x=0, $y=0,
➥$width=0, $height=0,
➥$style="", $transform="");
function SvgLine($x1=0, $y1=0,
➥$x2=0, $y2=0, $style="",
➥$transform="");
```

2. Lista betu.php

```
<?

function szin()
{
   $veletlen = dechex(rand(0, 255));
   return strlen($veletlen) <2? "0" . $veletlen :
   ➥$veletlen;
}

function szinkod()
{
   return "#" . szin() . szin() . szin();
}

if($_POST["szoveg"])
{
   define("SVG_CLASS_BASE",
$_SERVER["DOCUMENT_ROOT"] . "/svg/");
   $betukoz = 8;
   $min_betumeret = 8;
   $max_betumeret = 18;
   $magassag = 30;
   require_once("Svg.php");
   $szoveg =$_POST["szoveg"];
   $hossz = strlen($szoveg);
   $svg =& new SvgDocument($betukoz *($hossz+1) ,
   ➥$magassag);
   for($i=0; $i<strlen($szoveg); $i++)
   {
      $svg->AddChild(new SvgText(($i+1)*$betukoz,
      ➥$magassag/2, substr($szoveg, $i, 1),
                        "font-size:".
➥rand($min_betumeret,$max_betumeret) .
➥";text-anchor:middle; fill:" . szinkod()));
   }
   $svg->printElement();
}
else {
   header("Content-type: text/html;
   ➥charset=iso8859-2");
   ?><!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
   ➥Transitional//EN">
   <html>
   <head>
     <meta
   content="text/html; charset=ISO-8859-2"
   http-equiv="content-type">
   <title>SVG</title>
   </head>
   <body>
   <form method="post"
action="<?$PHP_SELF?>"><input
   name="szoveg"><input
   type="submit"></form>
   </body>
   </html>
<?
}
?>
```

2. ábra A betu.php futási eredménye Konquerorban

```
function SvgEllipse($cx=0,
➥$cy=0, $rx=0, $ry=0,
➥$style="", $transform="");
function SvgCircle($cx=0,
➥$cy=0, $r=0, $style="",
➥$transform="");
```

Ezekhez a függvényekhez sem szükséges megadni az összes itt felsorolt paramétert. Ne feledjük, hogy az `AddChild`-ot a példaprogramhoz hasonlóan használni kell a létrejött grafikai elemekre. Figyelem! A *PHP*-s program a bemenetet semmilyen módon nem ellenőrzi. Ez egy valódi alkalmazásban megengedhetetlen. Tanulságos eset történt a cikk írása közben, kapcsolódik a bevezetőben említett nyílt forrás – könnyű hibakeresés témához. Szokásomhoz híven a *PHP* és a *HTML* kódot *UTF-8*-as karakterkészlettel készítettem, de a program egyáltalán nem akart helyesen működni ékezetes karakterekkel. Ekkor megnyitottam a rendelkezésemre álló *php* fájlt (az *SvgDocument.php*-t) és megnéztem az objektum belső kódolását. Akár egy mozdulattal át is cserélhettem volna *UTF-8*-ra, de azt szerettem volna, ha az objektum módosítás nélkül felhasználható legyen.

Merre tovább?

Ez a két egyszerű példaprogram és a magyarázatuk azt a célt szolgálja, hogy az olvasó ráérezzen arra, hogy a szerveroldali *SVG* gyártás miféle lehetőségekkel bír. Akik jártasak *PHP*-ben és *Perl*-ben, azok tudják, hogy mindkét nyelvnek nagy erőssége az adatbázis-elérés, adatfeldolgozás. Az adatbányászat egyik eleme lehet az adatbázis megjelenítése *SVG* segítségével. Időjárás-ikon készíthető úgy, hogy egy szkript feldolgoz egy időjárás *RSS*-csatornát (➲ *www.rssweather.com*) és még sorolhatnánk. Egyszerű eszközökkel látványos megoldásokhoz juthatunk. A sorozat ugyan ennél a pontnál véget ér, de az *SVG* lehetőségei természetesen nem.

Novák Áron
(aaron@szentimre.hu)
BME-VIK-es gólya, műkedvelő rendszergazda. Jelenleg leginkább a NetBeans-szel és mindenféle hordozható eszközzel foglalkozik, legalábbis mindazokkal amelyeket meg lehet szólaltatni Linux alatt.