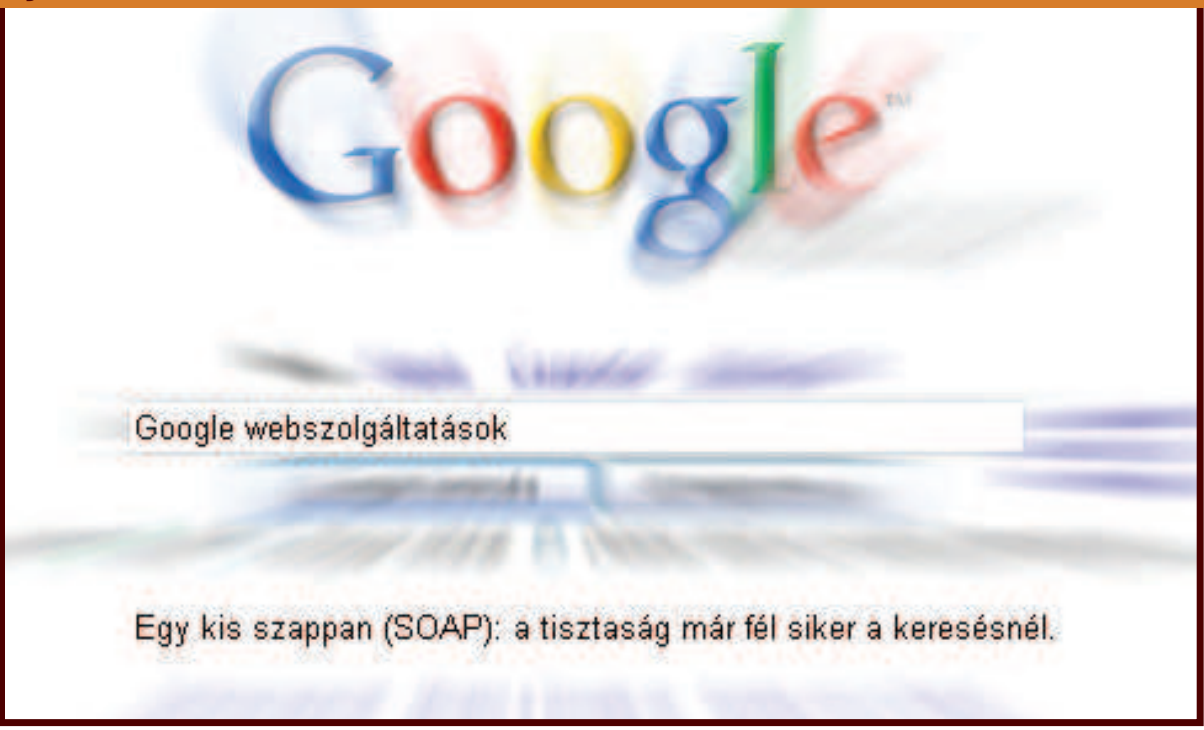

Az elmúlt hónapokban több olyan az ***Amazon*** által rendelkezésre bocsátott webszolgáltatását vizsgáltunk, amelyek lehetővé teszik, hogy viszonylag könnyedén keressünk annak katalógusában. Az ***Amazon*** jó néhány évvel ezelőtt úgy döntött, hogy a webszolgáltatásainak nagy részét ingyenessé teszi, azt feltételezve, hogy így többen lesznek azok, akik végül vásárolnak a webhelyükről. Valóban, ma már rengeteg fejlesztő használja az ***Amazon*** webszolgáltatásait különböző dolgok létrehozására, az egyedi könyvesboltoktól kezdve a könyvesbolt-kezelést segítő programokig.

Az ***Amazon*** nem az egyetlen kereskedelmi webhely, amely megnyitotta a katalógusát a külvilág számára. Az internet másik ***King Kongja***, a ***Google*** néhány évvel ezelőtt szintén közzétette a web ***API***-jait (alkalmazásprogramozó illesztőfelület). Ezek az ***API***-k lehetővé teszik a ***Google*** terjedelmes webes katalógusában történő keresést. Lehetetlen megmondani, hogy a világ legnagyobb katalógusáról van-e szó, de véleményem szerint ez nem annyira lényeges. A ***Google*** katalógusa elég nagy és elég gyakran frissül ahhoz, hogy az esetek döntő többségében azt használjam elsődleges keresőmotorként.

A ***Google*** az utóbbi néhány évben számos különböző ***API***-t tett elérhetővé. Ebben a hónapban a legegyszerűbbet vizsgáljuk, amely a webes archívumban történő alapvető keresésekre szolgál. Megvizsgáljuk, hogy a ***Google*** hogyan reklámozza a webszolgáltatásait a ***WSDL (Web Service Description Language – Webszolgáltatás-leíró nyelv)*** segítségével, és hogy miként indíthatunk olyan ***SOAP*** hívásokat, amelyekkel a saját céljainknak megfelelő kereséseket végezhetünk a ***Google*** hatalmas könyvtárában.

Az első lépések

A ***Google API***-k első lépései nem tartogatnak különösebb meglepetést azok számára, akik dolgoztak már az ***Amazon*** webszolgáltatásaival. Kezdéskor mindkét cég elvárja a regisztrációt a szolgáltatások használatához. A – mindkét esetben ingyenes – regisztráció során kapunk egy azonosítót, amely a kiszolgálóhoz küldött összes kérésben szerepel majd. Ahhoz, hogy ***Google*** kulcsot szerezzünk, először ***Google*** fiókot kell nyitnunk. Nos, egy ideje már rendelkezem „Google fiókkal”, amelyet a ***Gmail*** illetve a testreszabott híroldal szolgáltatáshoz használok. Úgy tűnik azonban, hogy az ***API***-k egy másik fiókcsoporthoz tartoznak.

Kissé furcsának találtam, a „fő” Google fiókba történő bejelentkezés után az ***API*** rendszerben még külön regisztrációra és bejelentkezésre van szükség.

Mindezek ellenére a fiók létrehozása egyszerű és magától értetődő. Menjünk a ***Google API***-k főoldalára (➲ ***www.google.com/apis***), kattintsunk a ***„create Google account”*** (***Google*** fiók létrehozása) szövegre, majd töltsük ki az űrlapot. A ***HTML*** űrlap elküldése után nem sokkal a fiók létrehozását visszaigazoló e-mailt kapunk a ***Google***-től, amely tartalmazza a ***Google*** kulcsot, valamint a fiók létrehozásának megerősítéséhez szükséges ***URL***-t. A fiók létrehozásának jóváhagyása után máris elkezdhetjük használni a ***Google*** kulcsot, és olyan programokat készíthetünk, amelyek kihasználják a ***Google*** webszolgáltatásainak előnyeit.

Azonban mielőtt ezt megtennénk, érdemes megfontolnunk, hogy a ***Google*** milyen módon korlátozásokat alkalmaz a szolgáltatásnál, és hogy milyen adatokat kapunk azon keresztül. Az ***Amazon*** másodpercenként csak egy ***API*** hívást engedélyez a tagoknak, ami egy adott 24 órás időtartamot tekintve legfeljebb 86400 hívás. Ezzel szemben a ***Google*** 24 óránként legfeljebb 1000 hívást tesz lehetővé a felhasználók számára.

Ezen felül a határértékek meghatározásának módja jelzi, hogy miként kezelik a korlátok túllépését.
A *Google* akkor ad vissza hibaüzenetet, ha az elmúlt 24 órában több, mint 1000 lekérdezést végeztünk, míg az Amazon csak olyankor jelez, amikor egy lekérdezés ugyanabban a másodpercben érkezik, mint az előző.
A hibaüzenet küldését megelőzően egyik szolgáltatás sem figyeli ezeket a számokat, az Amazon korlátozásának túllépését azonban nyilvánvalóan könnyebb orvosolni (egy másodperc felfüggesztéssel, majd újbóli próbálkozással), mint a *Google*-ét (mivel előfordulhat, hogy a programot akár 24 óráig fel kell függeszteni, mielőtt újra próbálkoznánk).
Jogi szempontból a két webhely szolgáltatásai számos ponton különböznek. Az *Amazon* állt elő elsőként azzal az ötlettel, hogy társul a webes kereskedőkkel, az adatbázisához kapcsolódó kereskedelmi szolgáltatások létrehozására buzdítva az embereket. Ezzel szemben a *Google* kifejezetten megtiltja a felhasználóknak, hogy a keresési találatokra építve kereskedelmi szolgáltatást hozzanak létre. (Ha valaki internetes keresési adatokra épülő kereskedelmi szolgáltatást szeretne létrehozni, érdemes megtekinteni az *Amazon Alexa Web* keresőplatform szolgáltatását, amely nem alkalmaz ilyen korlátozásokat. Ugyanakkor 1000 kérésenként 25 centet kell fizetni, ami egy népszerű webhelyen hamar megsokszorozódhat.)
Végül, a két webhely között technikai eltéréseket is tapasztalhatunk. Az *Amazon API*-jai *SOAP*-on és *REST*-en keresztül egyaránt működnek, így a fejlesztők választhatnak a két formátum közül. A *Google* viszont csak *SOAP* felületet biztosít a keresőmotorjához, tehát a keresőrendszer létrehozásához *SOAP* ügyfélkönyvtárat kell telepítenünk és alkalmaznunk. Szerencsére a legtöbb nyelv rendelkezik olyan magas szintű könyvtárakkal, amelyek lehetővé teszik a *SOAP* hívásokat.

SOAP::Lite

A *SOAP* – amelyet korábban *Simple Object Acces Protocol*-nak (*egyszerű objektum-hozzáférési protokoll*) neveztek, a betűszó azonban ma már hivatalosan nem jelent semmit – segítségével viszonylag könnyen küldhetünk *XML*-be tokozott lekérdezéseket a kiszolgálónak. Ezután a kiszolgáló *XML*-ként kódolt választ küld vissza. Az évek során a *SOAP* messzire kalandozott az egyszerű gyökereitől. Annak ellenére, hogy a *SOAP* még mindig könnyebben érthető, megvalósítható és alkalmazható, mint bizonyos bonyolultabb protokollok (például a *COBRA*), sokkal nehézkesebb, mint amennyire azt a legtöbben elismerik. Amennyiben lehetséges, én személy szerint jobban szeretek *XML-RPC*-t használni a webszolgáltatásokhoz. Az *XML-RPC* ugyan nem biztosítja a *SOAP* összes szolgáltatását, de sokkal könnyebb vele dolgozni.
Mindenesetre a *Google* a *SOAP* használatát várja el, és mivel manapság számos jó *SOAP* ügyfélkönyvtár létezik, nem kell, hogy féljünk a használatától. A *Perl* programozóknak egy különösen erős megvalósítás áll rendelkezésére, amelyet *SOAP::Lite*-nak hívnak. A cikk programozási példáiban a *Perlt* és a *Soap::Lite*-ot alkalmazzuk. Fontos, hogy a modulnév *Lite* (könnyű) része azt mutatja, hogy a programozók mennyire könnyen megvalósíthatják a webszolgáltatásokat, és nem a *SOAP* lebutított változatát jelöli. A következő paranccsal telepíthetjük a *SOAP::Lite* legfrissebb változatát a *CPAN*-ból:

```
perl -MCPAN -e 'install
➥SOAP::Lite'
```

A *SOAP::Lite* telepítés megkérdezi, hogy milyen ellenőrzéseket szeretnénk végezni a modul telepítése előtt, ha egyáltalán akarunk tesztet futtatni. Én általában elfogadom az alapbeállításokat, de az igényeinknek megfelelően esetleg néhányat hozzáadhatunk vagy eltávolíthatunk.
A *SOAP::Lite* telepítése után ideje írnunk egy olyan programot, amelyhez *Google* szükséges. Ehhez azonban ismernünk kell a szolgáltatás *URL*-jét, azt, hogy milyen eljárást hívunk meg a *Google* számítógépén, valamint a küldeni kívánt összes paraméter típusát és nevét. Ezeket kézzel is megadhatjuk, de ez sok munkát jelentene számunkra. Ezen kívül a *Google* jelenleg az ➲ *api.google.com/search/beta2* címre várja a *SOAP* kéréseket. Ha a *Google* valaha figyelmeztetés nélkül megváltoztatja ezt az *URL*-t, valószínűleg sokan meglepődnek és bosszankodnak majd.
Szerencsére a *Google* rendelkezésre bocsátott egy *WSDL* fájlt, amely leírja a *Google API*-k nyújtotta szolgáltatásokat, valamint a rendszer által elfogadott kérés és válasz paramétereket. Emellett meghatározza a lekérdezések célállomását, ami (elméletben) lehetővé teszi a *Google* számára, hogy a fejlesztők előzetes értesítése nélkül változtasson a szolgáltatáson. Természetesen ez azt feltételezi, hogy maga a *WSDL* fájl azonos helyen marad, továbbá, hogy a szolgáltatások neve sem változik és azok mindegyike valahol dokumentált, mivel a meghívott eljárások kiválasztása továbbra is emberi beavatkozást igényel.
A *WSDL XML*-ben íródott, és meglehetősen könnyen érthető, ha felismerjük, hogy csupán egy adott kiszolgálón elérhető különböző webszolgáltatásokat jellemzi: a bemenetek számát, nevét és típusát írja le.
A `doGoogleSearch` *WSDL* bejegyzése így a következő:

```
<message name="doGoogleSearch">
<part name="key"
➥type="xsd:string"/>
<part name="q"
➥type="xsd:string"/>
<part name="start"
➥type="xsd:int"/>
<part name="maxResults"
➥type="xsd:int"/>
<part name="filter"
➥type="xsd:boolean"/>
<part name="restrict"
➥type="xsd:string"/>
<part name="safeSearch"
➥type="xsd:boolean"/>
<part name="lr"
➥type="xsd:string"/>
<part name="ie"
➥type="xsd:string"/>
<part name="oe"
➥type="xsd:string"/>
</message>
```

Ahhoz, hogy egy *SOAP::Lite*-ot használó *Perl* programon belül *WSDL*-t alkalmazzunk, a `SOAP::Lite->service` parancsot hívjuk meg az *WSDL* fájl

1. Lista google-query.pl

```
#!/usr/bin/perl
use strict;
use diagnostics;
use warnings;
use SOAP::Lite;
# --------------------------------
# Get the Google key from ~/.google_key
my $google_key_file = "/Users/
➥reuven/.google_key";
open GOOGLE_KEY, $google_key_file or die
➥"Cannot read
'$google_key_file': $! ";
my ($google_key) = <GOOGLE_KEY>;
chomp $google_key;
close GOOGLE_KEY;
# --------------------------------
# Get the command-line argument
if ($#ARGV != 0)
{
    print "$0: Invoke with a single argument,
    ➥your Google search term.\n";
    exit;
}
my $query_string = shift @ARGV;
# --------------------------------
# Get the WSDL file
my $google_wsdl = "http://api.google.com/
➥GoogleSearch.wsdl";
my $query = SOAP::Lite->service($google_wsdl);
# --------------------------------
# Use the WSDL to make the query
my $starting_page = 1;
my $max_results = 10;
my $filter = 'false';
my $geographic_restriction = '';
my $safe_search = 'false';
my $language_restriction = '';
my $results =
    $query->doGoogleSearch($google_key,
                $query_string,
                $starting_page,
                $max_results,
                $filter,
                $geographic_restriction,
                $safe_search,
                $language_restriction, 'utf-8',
                ➥'utf-8');
my @results = @{$results->{resultElements}};
if (@results)
{
    # Iterate through each result we got
    my $counter = 1;
    foreach my $result (@results)
    {
    print "Result $counter of ", $#results + 1,
    ➥":\n";
    foreach my $key (sort keys %{$result})
    {
        my $value = $result->{$key};
        # Is this a hash value?  If so,
        Vdisplay it accordingly
        if (UNIVERSAL::isa($value, 'HASH'))
        {
        print "\t'$key':\n";
        foreach my $subkey (sort keys
        ➥%{$value})
        {
            print "\t\t'$subkey' => '$value->
            ➥{$subkey}'\n";
        }
        }
        # Display the value as a simple string
        else
        {
        print "\t'$key' => '$value'\n";
        }
    }
    $counter++;
    }
}
else
{
    print "There were no results for your query
    ➥of '$query_string'.\n";
}
```

URL-jével. Ha a fájl a helyi fájlrendszerben található, ügyeljünk, hogy az ***URL*** előtt `file:` álljon. Például:

```
my $google_wsdl =
➥"http://api.google.com/
➥GoogleSearch.wsdl";
my $query = SOAP::Lite->service
➥($google_wsdl);
```

Ezután a ***SOAP::Lite*** elég okos ahhoz, hogy végignézze a ***WSDL***-t és dinamikusan elérhetővé tegye az összes hirdetett szolgáltatást, hogy a következőt tehessük:

```
my $results =
$query->doGoogleSearch
➥($google_key,
                $query_string,
                $starting_page,
                $max_results,
                $filter,
                $geographic_
                ➥restriction,
                $safe_search,
                $language_
                ➥restriction,
                ➥'utf-8',
                ➥'utf-8');
```

Mi történik itt? A ***WSDL***-ben meghatározott bemenetek és a `$query->doGoogleSearch()`-nek átadott paraméterek egy az egyben leképezhetők egymással.

Egyszerű keresések a doGoogleSearch segítségével

Láttuk tehát a *Perlben* írott *Google* keresőprogram magját. Már csak annyi van hátra, hogy áttekintsük a bemeneti paramétereket és a `$results` tartalmát, amelyben a *Google* által visszaadott találatok szerepelnek.

Az *API* ➲ *www.google.com/apis/reference* címen található dokumentációja leírja a bemeneti paramétereket. Mindegyik kötelező, de vannak olyanok, amelyek fontosabbak, mint a többi: a *Google kulcsot* (`google_key`) és a lekérdezés-karakterláncot (`query_string`) általában megadjuk, a többit pedig egyszerűen alapértékkel adjuk meg, ahogy azt az *1. Listában* láthatjuk.

Az emberek többsége (magamat is beleértve) általában a lehető legtöbb weboldalt akarja keresni a lekérdezésekkel, mégis vannak olyan esetek, amikor célszerűbb, ha csak egy bizonyos terület vagy nyelv kiszolgálóiról gyűjtünk adatokat. Mivel a *Google* ezt lehetségessé, sőt, magától értetődővé teszi, számos különféle érdekes alkalmazás előtt nyílik meg az út.

Ahogy *SOAP*-ban kódolt *XML*-ként küldjük a lekérdezést a *Google*-nak, ugyanúgy, *SOAP*-ban kódolt *XML*-t kapunk eredményként. Mivel azonban a *SOAP::Lite* megkímél bennünket attól, hogy akár egy hajszálnyi *XML*-t is kelljen írnunk a lekérdezéshez, a válasz hasonlóképpen elszigetelődik. A `$results` változó *Perl* kezelőfelületet biztosít a válaszként kapott adatokhoz.

Pontosan milyen adatokat is kapunk? Ahhoz, hogy ezt megtudjuk, ismét meg kell néznünk a *WSDL* fájlt. Többek között azt mutatja, hogy az alábbihoz hasonló eredmények halmazaként kapjuk meg a válaszokat:

```
<xsd:complexType name=
➥"ResultElement">
<xsd:all>
    <xsd:element name="summary"
    ➥type="xsd:string"/>
    <xsd:element name="URL"
    ➥type="xsd:string"/>
    <xsd:element name="snippet"
    ➥type="xsd:string"/>
    <xsd:element name="title"
    ➥type="xsd:string"/>
    <xsd:element name=
    ➥"cachedSize" type=
    ➥"xsd:string"/>
    <xsd:element name=
    ➥"relatedInformationPresent"
    ➥type="xsd:boolean"/>
    <xsd:element name=
    ➥"hostName" type=
    ➥"xsd:string"/>
    <xsd:element name=
    ➥"directoryCategory" type=
    ➥"typens:
    ➥DirectoryCategory"/>
    <xsd:element name=
    ➥"directoryTitle" type=
    ➥"xsd:string"/>
</xsd:all>
</xsd:complexType>
```

Más szóval, a *Google*-től visszakapott eredmények (legfeljebb tíz) mindegyike biztosítja az összes olyan információt, amely a *Google* találati oldalával megegyező oldal létrehozásához szükséges. Ezen felül kiválaszthatjuk és meghatározhatjuk a megjeleníteni kívánt elemeket, amelyek például csak a címet és a `dmoz` könyvtárkategóriát illetve címet mutatják. Azt is megtehetjük, hogy a keresett oldal egy rövid részletét jelenítjük meg. Esetleg a felsoroltak közül az összeset. Vagy egyiket sem.

A `doGoogleSearch` nem az egyetlen *WSDL* fájl által leírt eljárás. Számos más eljárás is létezik, amelyek például a *Google* tárolt oldalait kezelik, illetve az egyes szavak helyesírását ellenőrzik. Amikor a webszolgáltatásokat elérhetővé tették a nyilvánosság számára, gyakran említették példaként, hogy a szövegszerkesztők így távoli webszolgáltatást hívhatnak meg a helyesírás-ellenőrzéshez, és nem kell beépített rendszerrel felszerelni azokat. Erre még sokat kell várnunk, de elképzelhető, hogy a *Google API*-ját használjuk egy ilyen szolgáltatás kísérleti változatánál.

Mindezek mellett a kimeneteket bemenetként használhatjuk egy másik webszolgáltatás-hívásnál, helyben és távolról egyaránt. Egyre népszerűbb az a gyakorlat, amelynek során több webhelyről származó adatokat vegyesen alkalmaznak, különösen a *Google* térkép *API*-jai esetében. Bámulatos látni, hogy mi minden történik amikor ilyen módon kombináljuk a szolgáltatásokat – ezt a következő hónapokban vizsgáljuk majd.

Összegzés

Ebben a hónapban röviden áttekintettük a *Google* kereső *API*-ját. Sikerült létrehozni a *Google* keresőoldalának parancssor-változatát, olyan egyszerű eszközök segítségével, mint a *Perl SOAP::Lite* modulja.

A következő hónapokban a *Google* térkép *API*-jával foglalkozunk majd, és kiderül, hogyan készíthetünk vegyes szolgáltatásokat, amelyek többféle adatforrást ötvöznek.

Linux Journal 2006., 145. szám

Reuven M. Lerner régóta web illetve adatbázis tanácsadó, jelenleg az Illinois állambeli Evanstonban található Northwestern University PhD hallgatója, Learning Sciences szakon. A feleségével nemrég ünnepelték Amotz David fiuk születését.

KÓDOK

A cikkben szereplő kódok a ➲ ftp.scc.com/pub/lj/listings/issue145/8866.tgz címen találhatók.

KAPCSOLÓDÓ CÍMEK

➲ www.linuxjournal.com/article/8881