Cikkgyűjtemények tartalmának begyűjtése

Adjunk lehetőséget alkalmazásunknak, hogy kedvenc weblapjainkról összegyűjtse a friss írásokat!

Az elmúlt néhány hónapban az *RSS* és *Atom XML*-alapú fájlformátumokkal foglalkoztunk, azzal a párossal, mellyel könnyedén elkészíthetjük és elterjeszthetjük egy weboldal összefoglalóját. Igaz ugyan, hogy – mint az ismeretes – az ilyesfajta cikkgyűjtemények készítése hagyományosan inkább a Weblogokra és híroldalakra jellemző, egyre nagyobb az érdeklődés más területeken is. Bármilyen web alapú információforrás esetében érdekes és hasznos jelölt lehet akár az *RSS* akár az *Atom*.

Ez idáig azt néztük meg, hogyan készíthetünk *RSS* és *Atom* tartalmat Weblapunkhoz. Természetesen az cikkgyűjtemény tartalom elkészítése csak az egyenlet fele. Legalább ilyen fontos és talán még hasznosabb megérteni, hogyan szerezhetjük be és használhatjuk ezeket az cikkgyűjtemény tartalmakat saját webhelyünkről és más érdekes oldalakról.

Amint azt láthattuk, három különböző cikkgyűjtemény tartalom létezik: az ***RSS 0.9x*** és fejlettebb verziója az ***RSS 2.0,*** a nem együttműködő ***RSS 1.0***, valamint az ***Atom***. Valamennyi nagyjából azonos feladatot végez el és elég nagy mértékű átfedés van a szabványok között. Ugyanakkor, a hálózati protokollok általában nem működnek valami fényesen, ha naivan feltételezzük, hogy minden rendben lesz és elegendően azonos működésű. Ez alól az cikkgyűjtemények sem kivételek. Amennyiben valamennyi összesítővel rendelkező lapot el szeretnénk tudni olvasni, az összes protokollt és azok verzióit meg kell értenünk. Az *RSS* például jelenleg kilenc különféle verzióval rendelkezik amelyhez ha az *Atom*-ot is hozzávesszük, összesen tíz különféle cikkgyűjtemény formát használhat valamely weblap. A legtöbb eltérés valószínűleg elhanyagolható, de nem lenne bölcs dolog teljesen figyelmen hagyni őket, vagy feltételezni, hogy mindenki a legfrissebb verziót használja. Ideális esetben van valamilyen modulunk vagy eszközünk amely több különféle protokoll nyelvén is ért eltüntetve a különbségeket, miközben valamennyi protokoll egyedi előnyeit kihasználja.

E hónapban ***Mark Pilgrim univerzális tartalom értelmezőjét (Universal Feed Parser)*** vesszük szemügyre, amely e problémára kínál nyílt forráskódú megoldást. ***Pilgrim*** jól ismert weblog szerző és ***Python*** programozó, valamint az ***Atom*** cikkgyűjtemény formátum megalkotásának egyik kulcsfigurája. Ez persze nem is meglepő, ha azt vesszük mennyi problémába ütközött az ***Universal Feed Parser*** megíráskor. A program a ***Microsoft*** üzleti ***CDF*** formátumát is kezeli, amely az aktív asztal és programfrissítések összesítőit terjeszti. A ***Linux*** asztali felhasználók számára ez a rész talán nem túl izgalmas viszont érdekes egységesítési lehetőséget kínál a már telepített ***Microsoft*** rendszerekkel. Az ***Universal Feed Parser*** jelenleg a 3.3 verziónál tart, saját kategóriájában a legjobbnak számít nyelvtől és licensztől függetlenül.

A feedparser telepítése

A ***feedparser*** telepítése rendkívül egyszerű. Töltsük le a legfrissebb verziót, tegyük a telepítési könyvtárába és gépeljük be a

```
python setup.py installc
```

parancsot. Ezzel elindítjuk a ***Python*** szabványos telepítési eszközét, amely a ***feedparser***-t ***a Python site-packages*** könyvtárába helyezi. A ***feedparser*** telepítése után próbáljuk ki a ***Python*** segítségével valamelyik héjablakból:

```
>>> import feedparser
```

A >>> jelek az interaktív módban meghívott ***Python*** szabványos parancssorának felelnek meg. A fenti utasítás beolvassa a ***feedparser*** modult a ***Pythonba***. Ha nem telepítettük a ***feedparser***-t, vagy valamilyen gond történt telepítés során, a parancs végrehajtása ***Python ImportError*** hibát okoz.

Miután importáltuk a modult a memóriába, nézzünk belea legfrissebb hírekbe a ***Linux Journal w***eblapján. Gépeljük be:

```
>>> ljfeed = feedparser.parse
("http://www.linuxjournal.com/news.rss")
```

Nem szükséges megadnunk milyen protokoll vagy verzió szerinti tartalmat szeretnénk az értelmezővel feldolgoztatni – a csomag elég okos ahhoz, hogy magától felismerje a verziókat, még olyankor is amikor az *RSS* tartalom maga nem tudja beazonosítani a verzióját. Írásunk születésekor a LJ lapja ***PHPNuke*** alapú az tartalmat használ, melyet egyértelműen ***RSS 0.91*** típusként sikerült azonosítani.

Miután letöltöttük az új tartalmat, megnézhetjük pontosan hány bejegyzést kaptunk, amit többé kevésbé a kiszolgáló beállításai határoznak meg:

```
>>> len(ljfeed.entries)
```

természetesen, az elemek száma kevésbé érdekes mint az elemek maguk, ezeket egy egyszerű `for` ciklussal kaphatjuk meg:

```
>>> for entry in ljfeed.entries:
...     print entry[´title´]
...
```

Ne feledjük, a ciklus sorait beljebb kell kezdenünk, hogy a ***Python*** tudja mely sorok tartoznak a ciklusba. Aki még csak most ismerkedik a ***Python*** nyelvvel, esetleg furcsának találja a `...` jelzéssel kezdődő sorokat amelyek azt mutatják, hogy a ***Python*** készen áll és a `for` ciklus utáni sorokra várakozik. A ciklus végrehajtásához egyszerűen csak nyomjunk ENTER-t és máris láthatjuk az összes friss címet.
Csinos átnézeti képet kaphatunk az ***URL***-ekről és a címekről a ***Python*** karaktersorozat behelyettesítésének segítségével:

```
>>> for entry in ljfeed.entries:
...     print ´<a href="%s">%s</a>´ % \
...     (entry[´link´], entry[´title´])
```

Mint korábban jeleztem, a `feedparser` megpróbálja feloldani a különböző verziók közötti különbségeket, és úgy dolgozhatunk a különböző összesítési formákkal mintha azok nagyjából azonos módon működnének. Ezért a fenti parancsokat saját weblogom cikkgyűjtemény tartalmára is megismételhetem. Nemrég tértem át a ***WordPress***-re, amely ***Atom*** tartalmat használ:

```
>>> altneufeed = feedparser.parse(
... "http://altneuland.lerner.co.il/wp-atom.php")
>>> for entry in altneufeed.entries:
...     print ´<a href="%s">%s</a>´ % \
...     (entry.link, entry.title)
```

Figyeljük, meg hogy ez az utóbbi példa az `entry.link` és `entry.title` attribútumokat használja, míg a korábbi példánk az `entry['link']` és `entry['title']` könyvtárhivatkozásokat alkalmazta. A ***feedparser*** rugalmasan próbálja megközelíteni a problémát, és egyazon információhoz többféle elérési lehetőséget kínál, kiszolgálva a különböző igényeket és stílusokat.

Milyen friss a hír?

A hírösszesítők és más ***RSS*** vagy ***Atom*** használó alkalmazások fő célja a nemrégiben frissített hírek összegyűjtése és bemutatása. Az hírösszesítő csak azokat a híreket szolgáltathatja amelyeket a kiszolgáló felajánl. Amennyiben az ***RSS*** tartalom csak a két legutóbb frissített elemet tartalmazza, az hírösszesítő felelőssége begyűjteni, gyorstárazni, és megjeleníteni azokat az elemeket, amelyek már nem kerülnek az összesítőkbe.

Ez két eltérő, bár szoros kapcsolatban álló kérdést vet fel: Hogyan biztosíthatjuk, hogy az összesítő csak olyan elemeket mutasson, amelyeket még nem láttunk? Van-e lehetőség rá, hogy az hírösszesítőnk csökkentse a terhelést a weblog kiszolgálón, és csak azokat a bejegyzéseket töltse le amelyek a legutóbbi frissítésünk óta kerültek fel? Az első kérdésre a válasz, hogy minden egyese elemnél ellenőriznünk kell a módosítási dátumot, amennyiben az létezik.
A második kérdés jelenleg egyre népszerűbb vitatéma a webes közösség berkeiben. Ahogy a weblog népszerűsége növekszik, úgy nő az egyes cikkgyűjteményekre feliratkozó emberek száma is. Ha a weblog cikkgyűjtemény tartalma 500 feliratkozót szolgál ki, és az összes feliratkozott óránként gyűjti be a frissítéseket, az óránként további 500 lekérdezést jelent amit a Webkiszolgálónak teljesíteni kell. Ha a cikkgyűjtemény tartalom a teljes webhelyet lefedi ez igen komoly elpazarolt sávszélességet is jelenthet – ami csökkenti az oldal válaszidejét más látogatók számára, illetve előfordulhat, hogy az oldal tulajdonosának fizetnie kell a engedélyezett havi sávszélesség túllépése miatt.
A ***feedparser*** használatával önmagunkhoz és az összefoglaló kiszolgálóhoz is kíméletesek lehetünk, ugyanis lehetővé teszi, hogy csak a akkor töltsük le az cikkgyűjtemény tartalmakat, ha van figyelemre méltó újdonság. Ez úgy lehetséges, hogy a ***HTTP*** modern verziói lehetővé teszik, hogy a kérelmező ügyfél beállítsa az ***If-Modified-Since (adott időpont óta módosult)*** fejlécet, amit a dátum követ. Amennyiben a kért ***URL*** módosult a megadott dátum óta, a kiszolgáló az ***URL*** tartalmával válaszol. Ha viszont a kért ***URL*** változatlan, a kiszolgáló a 304-es válaszkódot küldi, jelezve, hogy az aktuális tartalom továbbra is a korábban letöltött verzió.
Mindezt úgy érhetjük el hogy egy lehagyható módosító paramétert adunk át a `feedparser.parse()` függvénynek. Ez a paraméter a ***time*** modul szerint megadott szabványos ***Python*** szerkezet, melyben az első hat elem az év, hónap szám, nap szám, óra, perc és a másodperc. Az utolsó három elem minket jelenleg nem érdekel így nyugodtan hagyhatjuk őket 0 értéken. Tehát, amennyiben a 2004 Szeptember 1. óta beérkezett tartalmakat akarjuk látni, a következőket gépeljük be:

```
last_retrieval = (2004, 9, 1, 0, 0, 0, 0, 0, 0)
ljfeed = feedparser.parse(
          "http://www.linuxjournal.com/news.rss")
```

Ha a ***Linux Journal*** kiszolgálója helyesen van beállítva a fenti kód a teljes cikkgyűjtemény tartalmat letölti az ***ljfeed***-be és vagy a `HTTP OK` állapot üzenetet kapja vissza (melynek számszerű kódja 200) vagy egy jelzést, miszerint a az adat nem változott az utolsó letöltés óta (amelynek számszerű kódja 304). Igaz ugyan, hogy ha tárolni szeretnénk mikor kértük le az utolsó cikkgyűjtemény tar talmat, több bejegyzést kell nyilvántartanunk, mégis fontos betartanunk ezt a szabályt, különösen ha rendszeresen kérünk tartalom frissítéseket, ugyanis ellenkező esetben alkalmazásunk nem igazán lesz szívesen látott vendég bizonyos honlapokon.

1. lista aggregator.py

```
#!/usr/bin/python

import feedparser
import sys

# ------------------------------------------------
# a személyes tartalom kimeneti állomány
# megnyitása

aggregation_filename = “myfeeds.html”
max_title_chars = 60

try:
    aggregation_file =
open(aggregation_filename,”w”)
    aggregation_file.write(“””<html>
<head><title>My news</title></head>
<body>”””)
except IOError:
    print “Error: cannot write ´%s´ “ % \
    aggregation_filename
    exit

# ------------------------------------------------
# A feeds.txt minden nem üres sora
# tartalom forrás.

feeds_filename = “feeds.txt”
feeds_list = []

try:
    feeds_file = open(feeds_filename, ´r´)
    for line in feeds_file:
        stripped_line = line.strip().rstrip()

        if len(stripped_line) > 0:
            feeds_list.append(stripped_line)
            sys.stderr.write(“Adding feed ´” + \
            stripped_line + “´\ n”)

    feeds_file.close()

except IOError:
    print “Error: cannot read ´%s´ “ %
    ➥feeds_filename
    exit

# ------------------------------------------------
# végiglépdelünk a feeds_list listán,
# és leszedjük a megfelelő tartalmat

for feed_url in feeds_list:
    sys.stderr.write(“Checking ´%s´...” %
    ➥feed_url)
    feed = feedparser.parse(feed_url)
    sys.stderr.write(“done.\ n”)

    aggregation_file.write(´<h2>%s</h2>\ n´ % \
                           feed.entries[0].title)

    # Végiglépdelünk az tartalom összes
    # bejegyzésén, megjelenítjük és beírjuk
    # az összefoglalóba
    for entry in feed.entries:
        sys.stderr.write(“\ tWrote: ´%s´” % \

entry.title[0:max_title_chars])

        if len(entry.title) > max_title_chars:
            sys.stderr.write(“...”)

        sys.stderr.write(“\ n”)

        aggregation_file.write(
           ´<li><a href=”%s”>%s</a>\ n´ %
           (entry.link, entry.title))

    aggregation_file.write(´</u2>\ n´)

# ------------------------------------------------
# HTML befejezése

aggregation_file.write(“””</body>
</html>
“””)
aggregation_file.close()
```

Munka a tartalmakkal

Most már van némi elképzelésünk róla, hogyan kell dolgozni a ***feedparser***-el, készítsünk hát egy egyszerű összesítő eszközt. Eszközünk bemenetét a ***feeds.txt*** állományból kapja kimenetét pedig a ***feeds.html HTML*** állományban adhatjuk meg. A programot a `cron` eszközzel futtatva és naponta belenézve a ***HTML*** állományba egyszerű de működőképes hírtartalmat kaphatunk a minket leginkább érdeklő honlapokról.

A ***feeds.txt*** állomány a tényleges tartalmak ***URL***-jeit tartalmazz és nem a honlapok címét ahonnét az abrakot le szeretnénk tölteni. Más szóval a felhasználóra bízzuk, hogy megtalálja és bevigye minden egyes tartalom ***URL*** címét.

A kifinomultabb összesítő eszközök általában képesek beazonosítani a tartalom ***URL*** címét a a webhely honlapjának fejlécében található `link` tag alapján.

Ráadásul, a korábbi figyelmeztetésem ellenére, miszerint minden hírösszesítőnek követnie kellene a legfrissebb híreket hogy ne terhelje túl a kiszolgálókat, ez a program nem tartalmaz ilyen képességet, hiszen megpróbáltam a tömörségre és olvashatóságra törekedni.

Az *1. listában* olvasható ***aggregator.py*** nevű program négy részre bontható:

1. Először is megnyitjuk a kimeneti fájlt, azaz a ***myfeeds.html*** nevű ***HTML*** formátumú szöveges állományt. Ezt a z állományt Webböngészőben való nézegetésre szánjuk.

Ezt a helyi, `file:///` ***URL***-el megadott állományt akár fel is vehetjük a személyes könyvjelzőink közé, vagy esetleg beállíthatjuk kezdőlapnak is. Miután meggyőződtünk róla, hogy valóban tudunk írni az állományba, megkezdjük a ***HTML*** állományt.

2. Beolvassuk a ***feeds.txt*** tartalmát, amelyben soronként egy tartalom ***URL*** található. Hogy elkerüljük a szóközök és üres karakterek használatából adódó problémákat, levágjuk az üres karaktereket és figyelmen kívül hagyunk minden sort amiben nincs legalább egy nyomtatható karakter.
3. Ezek után végiglépkedünk a `feeds_list` tartalom listán, és meghívjuk `feedparser.parse()` függvényt az adott ***URL***-re. Ha kapunk választ, kiírjuk a ***myfeeds.html*** kimeneti állományba, az ***URL***-el és a cikk címével együtt.
4. Végül lezárjuk a ***HTML***-t és az állományt.

A kódba belepillantva láthatjuk, hogy személyes használatra nagyon könnyen készíthetünk hírösszesítőt. Persze ez csak nagyon vázlatos alkalmazás. Ha használhatóbbá szeretnénk tenni, valószínűleg érdemes lenne a ***feeds.txt*** és ***myfeeds.html*** tartalmát relációs adatbázis-kezelőbe vinni, a tartalom ***URL*** címét pedig automatikusan vagy félautomatikusan meghatározni a lap ***URL*** címe alapján, továbbá érdemes lenne tartalom kategóriákat is alkalmazni, így a többszörös tartalmak egységes egészként lennének olvashatóak. Amennyiben a leírás ismerősen hangzik, a kedves olvasó biztosan ***Bloglines*** felhasználó, hiszen ez a web-alapú blog gyűjtögető éppen a fenti módon működik. Nyilvánvaló módon a ***Bloglines*** sokkal több tartalmat és sokkal több felhasználót kezel mint amennyi az egyszerű játékpéldánkban található. Ha el szeretnénk készíteni a ***Bloglines*** belső szervezeti verzióját, némi személyre szabott kóddal kiegészített ***Universal Feed Parser*** és egy adatbázis-kezelő (például a ***PostgreSQL***) alkalmazásával a kialakítás egyaránt könnyű és hasznos is.

Összefoglalás

A spanyolviasz feltalálásának esetét gyakran említik a számítógépipar problémájaként. ***Mark Pilgrim Universal Feed Parser*** programja lehet, hogy csak egy apró igényt elégít ki a programok világában, de ez az igény szinte bizonyosan növekedni fog, ahogy az cikkgyűjteményeket használó személyek és szervezetek száma egyre szaporodik. Amennyiben kedvünk támad cikkgyűjtemény tartalmakat olvasni vagy értelmezni, érdemes használnunk a ***feedparser***-t. Kipróbált és jól dokumentált alkalmazásról van szó, amely gyakran frissül és fejlődik, munkáját pedig gyorsan és jól végzi.

Linux Journal 2004. december, 128. szám

Reuven M. Lerner, veterán web/adatbázis tanácsadó és fejlesztő, aki jelenleg Northwestern Egyetem Learning Sciences programjának végzős hallgatója. Weblogja az altneuland.lerner.co.il címen olvasható, és a reuven@lerner.co.il címen érhető el.