tech.net

working with windows

III./01.szám
1344 Ft

HTML lite

A Wireless Markup Language

25. oldal

Ki mivel ♥?

Farkasokkal Netmonozó

8. oldal

MSDownload

– Patchwork

44. oldal

ISSN 1586518S
01
9 771586 518005

2002 Január

HTML lite

A Wireless Markup Language

Egy vérbeli informatikus mindenekelőtt hidegrázást kap a csigalassú, legfeljebb kétszintű grafikával (ha egyáltalán...) ellátott, 4-5 sorban, soronként 20 karakterben megjelenített szolgáltatásoktól, amit ráadásul rendes billentyűzet hiányában mindenféle trükkös módon lehet csak használni. Kínszenvedés az egész. A WAP halálra van ítélve, mondjuk kórusban. Aztán a kezünkbe kerül az első WAP-os telefon...

25. oldal

Farkasokkal táncoló

(III. rész)

Az előző alkalommal sikerült telepítenünk a fürtszolgáltatást, most megismerkedünk az adminisztrálására szolgáló mmc-modullal, a parancssori felülettel, a csoportokkal, a függőségekkel és az átköltözés rejtelmeivel.

4. oldal

Ki mivel ♥?

Farkasokkal Netmonozó

Emlékeznek még a ... típusú nyomtató esetére, mely beállt a fürtbe? Azt a cikket Lepenye Tamás követte el, a NetMon sorozatba „furakodva". Most pedig én fogok az ő szakterületébe belekotyogni egy fürtküzdelem kapcsán.

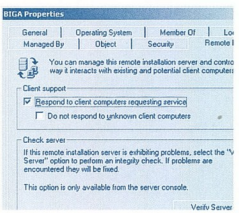
8. oldal

RIS

(III. rész)

A mai alkalommal a RIS kiszolgáló beállításait nézzük végig. A RIS összes felügyeleti szerve be van építve az Active Directory Users and Computers MMC snap-inbe, így ezzel a megszokott eszközzel *(szinte)* minden RIS-karbantartási feladatot elvégezhetünk.

10. oldal



A Windows NT és a Windows 2000 memóriakezelése

– más megközelítésben (olvasói levél)

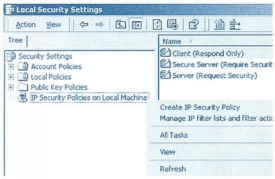
14. oldal

SPAM

– Jó vagy rossz?
Lehetséges válaszok

A spam mibenléte, az erre vonatkozó eddigi, jelenleg hatályos illetve a mindjárt hatályba lépő szabályozás alapos áttekintése megérdemel annyi energiát az olvasótól, amennyit az elolvasása igénybe vesz, mivel a jogi megközelítés változása igen tanulságos az Internet jogi szabályozásának egészét tekintve is.

16. oldal



Portszűrés IPSec-kel

Munkánk során sokszor kerülünk olyan helyzetbe, hogy egy kiszolgálót *(legyen a feladat web, levelezés vagy bármi más)*, meztelenül, pőrén kell az Internethez csatlakoztatni. Sehol egy jó kis tűzfal, marad az önvédelem. Az már alapszabály, hogy minden szolgáltatást, portot, amire nincs szükség, be kell zárnunk, de mégis, hogyan?

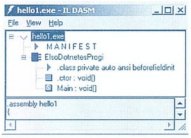
19. oldal

ASP Suli

Az Indexing Service

Az Indexing Service szolgáltatás - mely a háttérben csendben indexelgeti a dokumentumok tartalmát (hogy később azután segítségével kereshessünk közöttük) -megtalálható a Windows 2000 minden változatában. Ha a szolgáltatás fut, még az egyszerű keresés is felhasználja az Indexing Service-től kapott adatokat, de programozói felületének köszönhetően nagyszerűen használható scriptekből is.

21. oldal



.NET Akadémia

(I. rész)

Új év, új sorozat. Elérkezett az idő itt Magyarországon is, hogy belevágjunk egy ki tudja milyen hosszú ideig tartó kalandba, és feltérképezzük a .NET szövevényes világát. A .NET-ről, mint fejlesztési háttérről már írtam egy bevezetőt a Tech.net 2001. júniusi számban. Aki még egyáltalán nem találkozott a .NET-el, annak ajánlom a cikket elolvasásra, mert az abban leírtak ismeretét már alapul veszem ebben a részben.

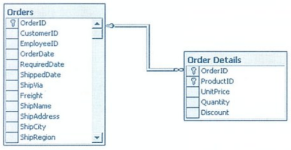
31. oldal

XMLgessünk

(IX. rész)

Az előző részben láttuk, hogyan kell használni az XmlReader és az XmlWriter osztályokat. Gyorsak, de eléggé alacsonyszintű interfészeket adnak a kezünkbe. Most megnézzük azokat a technológiákat, amelyek rájuk épülnek, és a segítségükkel így jóval fejlettebb módon is tudunk xml dokumentumokat manipulálni. A főszereplők a DOM, az XSLT és az XPath lesznek.

36. oldal



SQL Server

Join algoritmusok

Az elmúlt alkalommal megvizsgáltuk a Query Optimizer működésének főbb alapelveit, és kiveséztünk néhány egyszerű végrehajtási tervet *(query plan)*. Most – igéretemhez híven – a kapcsolási, avagy join stratégiák kerülnek sorra. Joggal kérdezhetik, hogy minek ez nekünk, hisz mindig minden automatikusan dől el az SQL Serverben. Ez igaz. De hogy **milyen döntés** születik automatikusan, azt súlyosan befolyásolhatják a mi korábbi tervezési döntéseink.

40. oldal

MSDownload

– Patchwork

2001 október közepétől a Netacademia Kft üzemelteti Magyarország egyik legnagyobb letöltőközpontját. A Microsoft Magyarország megbízásából elkészült oldalak az eddig megszokott tartalommal, de eltérő formában jelentkeznek. Az új megjelenés könnyű kereshetőséget, világos áttekinthető formát biztosít a látogatók számára. Az innen letölthető programok közül csemegézünk ebben az új rovatban.

44. oldal

Rendszerváltás

IPv6

Csak semmi aggodalom! Most nem hullanak fejek, nem cserélik le az utcatáblákat, és azt sem írják elő, hogy hány ágú, milyen színű csillagokkal díszíthetjük a karácsonyfát. Szerencsére másról, sokkal érdekesebbről és hasznosabbról lesz szó: az Internetről. Helyesebben annak gyakorlatilag egyeduralkodó protokolljáról, az Internet Protokollról *(IP)*, amelynek jelenleg a 4-es számú verzióját használjuk.

45. oldal

Design Time

Eljött a január, a fogadkozások és jóslatok hónapja. Idén nem jósolok, és nem is fogadkozom – a cím mást sejtet. Az újévi fogadkozások valójában nem mások, mint életünk átszervezésének feledésbe merülő kísérletei. Mi, a tech.net magazin szerkesztősége, fordítva csináljuk. Nem fogadunk meg semmit, ennek azonban látható jelei lesznek. Az ojság némiképp átszervezett dizájnja a mementó: valami megváltozott! De mi?

A körülöttünk lévő világ. Amerika megtanulta: a legjobb mentési stratégia sem ér semmit, ha hiányzik az az emberi munkaerő, amely a halott adatokba életet lehelne. Minek állítsuk vissza a WTC alá temetett adatokat, ha nincs, aki értelmezze? A nagy cégek többé soha nem fognak arra törekedni, hogy többezer dolgozójukat ugyanabba a buildingbe költöztessék.

Hogy mi köze ennek az informatikához? Fenntartja a fejlődést még recesszió közben is. Ide nekem a videokonferenciát, a nagy, és még nagyobb sávszélességű kapcsolatokat, hogy a földrajzilag szétszórt telephelyek dolgozói legalább virtuálisan együtt legyenek! Mi kell ehhez? A telephelykapcsolat legyen – költségtakarékos módon – Internetkapcsolat. Ha Internetkapcsolat, és biztonság, akkor virtuális magánhálózatok. Ha virtuális magánhálózatok, akkor titkosítás és hálózati ismeretek. Ha hálózati ismeretek, akkor IP, IPv6, útválasztó protokollok, RRAS. Ha titkosítás, akkor IPSEC, Kerberos, Encrypting File System, Public Key Infrastructure stb. Nem mondhatjuk, hogy a mi kis vindóz barátunk nem készült fel jó előre erre a változásra. Mondjuk ki: ez a recesszió nekünk, informatikusoknak még jól is jöhet. Persze csak azoknak, akik valamivel előrébb járnak, mint társaik, mert ha a recesszió eléri a hagyományos iparágakat, a hagyományos rendszergazdik sem lesznek védve. A kiemelt tudású szakemberek viszont igen.

Nem fogadunk meg semmit, ennek azonban látható jelei lesznek. Az ojság némiképp átszervezett dizájnja a mementó: valami megváltozott! De mi?

2001

Hadd számoljak be most a tech.net magazin elmúlt egy évéről. Tizenhárom szám jelent meg, összesen közel hatszáz oldalon. Előfizetői táborunk töretlen lendülettel növekedett, kiadott példányszámunk decemberre elérte a négyezret. Hogy ez Magyarországon mekkora szám, azt jól illusztrálja, hogy a közelebbről meg nem nevezett, ámde egyetlen számítástechnikai hetilap is ebben a példányszámban jelenik meg! Az eredeti gondolatmenet követésével nullára redukáltuk a töltelékinformáció mennyiségét:

- Megszűnt a hírek rovat, mert egy kéthónapos átfutású lapban ennek helye nincs. Van az Interneten annyi de annyi portál, melyek annyi de annyi haszontalan hírrel bombáznak minket!
- Megszűnt a külső lapokból átvett cikkek ócska hagyománya. Ma már minden cikket a szerkesztőség munkatársai írnak. Rá kellett jönnünk, hogy a külföldi cikkek annyira távol állnak a magyar „virtustól", hogy tripla munka van velük. Le kell fordítani, majd kiszedni belőle az eredeti mű tárgyi tévedéseit, és végül stílust adni neki. Mindeközben néhány esetben többet tudtunk az adott témáról a cikk eredeti szerzőjénél.
- A sorozatok szinte tankönyvjelleget kölcsönöznek a lapnak. Új előfizetőink nemegyszer visszamenőleg az összes számot megveszik. El sem tudom mondani, mennyire szívet melengető, amikor valaki még az irodánkba is kicaplat, hogy mielőbb elvigye a kék-ezüstöt.

A töltelékinformáció megszűntével azonban a lap – bármennyi pici humort fecskendezünk is bele – súlyos olvasmánnyá vált. Gondoljunk csak az XML sorozatra!

Tulajonképpen emiatt alakult át a lap belső felépítése olvastatóbb, zaklatottabb formátumúvá. A kék színárnyalatát pedig az „idők szavára hallgatva" változtattuk meg. It's XP time!

Fóti Marcell
marcellf@netacademia.net

tech.net
working with windows

Szerkesztőség
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net
Főszerkesztő-helyettes: **Fülöp Miklós**
mick@netacademia.net
Szerkesztőség címe:
1105 Budapest, Ihász utca 13.
Tel.: 263-2732
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@technetklub.hu

Kiadja és terjeszti a
NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.

NetAcademia Kft.
Terjesztési, előfizetési információ:
Tel.: 263-2732
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft
Példányszám: 4.000

Előfizethető megrendelőlevélben a szerkesztőségnél:
1105 Budapest, Ihász utca 13.
Fax: 261-7145
http://technet.netacademia.net/subs

Hirdetésfelvétel:
BÁRSONYKALAPÁCS MARKETING
VELVET HAMMER MARKETING
Az ütütő siker kovácsa.
Felelős: **Balogh Zoltán**
Tel.: 489-4665
Fax: 489-4660
info@velvethammer.hu
1027 Budapest, Fő utca 67. V. 1.
Grafikai tervezés, kivitelezés, nyomdai előkészítés:
Bársonykalapács Marketing
Művészeti vezető: **Balogh Zoltán**

Nyomda:
Cerberus Kft.
1066 Budapest, Lovag u. 14.
Felelős vezető: **Schmidt Gábor**

ISSN 1586-5185

Farkasokkal táncoló *(III. rész)*

Az előző alkalommal sikerült telepítenünk a fürtszolgáltatást, most megismerkedünk az adminisztrálására szolgáló mmc-modullal, a parancssori felülettel, a csoportokkal, a függőségekkel és az átköltözés rejtelmeivel.

A Cluster Administrator

A telepítés befejezése után a start menü „Administrative tools" csoportjában megjelenik a „Cluster Administrator" ikon, amely grafikus felületet nyújt a clustererőforrások kezelésére. Ha a kiszolgáló konzolja helyett előnyben részesítjük a saját munkaállomásunkat a rendszer konfigurálására, akkor telepíthetjük az Advanced Server CD-ről az \i386 könyvtárból az Adminpak.msi csomagot. Ez a Windows 2000 Server valamennyi szolgáltatásának kezelőprogramját telepíti Windows 2000 munkaállomásunkra. NT4-re a csomag sajnos nem telepíthető.

Első indításkor meg kell adnunk a cluster nevét, hogy csatlakozhassunk hozzá, de a lehetséges neveket fel is fedezi a modul, ha megkérjük rá. A következő alkalmakkor már automatikusan csatlakozik a megadott clusternévhez. Ha ezt nem szeretnénk, akkor a

```
cluadmin -noreconnect
```

vagy

```
cluadmin -norecon
```

kapcsolókkal kerülhetjük el a csatlakozást. Előfordulhat például, hogy a cluster – amely ugye virtuális – valamilyen oknál fogva nem érhető el, ekkor az egyes node-ok nevét is megadhatjuk, így a csatlakozás sikeres lesz. Ha tudni szeretnénk, hogy pontosan hogyan csatlakoztunk, elég ránézni az ablak címsorára: a clusternév*(clusternév)* azt jelzi, hogy a clustert magát céloztuk meg, a clusternév*(nodenév)* viszont azt, hogy az egyik állomáson keresztül érjük el a fürtöt. A clusternév*(.)* azt jelzi, hogy az egyik állomásról LPC hívásokkal dolgozik a bedolgozó modul. Normális működés közben ennek nincs jelentősége, hiba esetén viszont nem árt tisztában lenni a kapcsolódás miértjével.

A bedolgozómodul bal oldalán a clusternév szerepel a fa gyökereként. Ebből nyílnak az erőforráscsoportok, az összes erőforrás gyűjtőmappája, a cluster konfigurációs mappája, valamint a fürt állomásai.

A Cluster Administrator felülete

Az ablak jobb oldalán a mindenkori aktív konténer tartalmát láthatjuk, nagyon gyakran állapotinformációkkal fűszerezve. Ez fontos momentum. A rendszergazdai felület precízen és azonnal megjeleníti a cluster komponenseinek állapotát, ami alapján a teljes cluster állapotára következtethetünk. Nem sok ilyen állapot létezik, érdemes felsorolni őket:

Online *(működik)*: A clustererőforrás létezik és működik. Ezt az állapotot szeretjük leginkább.

Offline *(nem működik)*: Az erőforrás nem üzemel. Ez nem biztos, hogy hiba. Az erőforrások meghatározott sorrendben vehetnek fel állapotokat, valahogy úgy, ahogy azt az ábra mutatja.

Az erőforrások lehetséges állapotai

Látható, hogy az offline állapot korántsem csak hibát jelent, hanem egy átmenetet is az egyik online állapotból a másikba. Ez történik például akkor, amikor egy csoportot átmozgatunk egyik állomásról a másikra.

Online pending *(élesztés közben)*: Az előzőekből következően ez egy átmeneti állapot. Lehetséges azonban, hogy az erőforrás ebben az állapotban beragad, ami komoly hibára utalhat.

Offline pending *(lekapcsolás alatt)*: Az előző állapot tükörképe.

Failed *(sérült)*: Ez a tényleges hiba, a cluster nem képes elindítani a csoportot vagy erőforrást. Gyakorlatilag bármely állapotot követheti „sérült" állapot. A Cluster Administrator elég szigorú, és már akkor is ilyen állapotjelzővel „címkézi" az erőforráscsoportokat, ha egyetlen erőforrás nem online állapotban van. Ez „ergonómiai hiba", ha lehet így fogalmazni.

A cluster parancssori felülete

A Cluster Administrator nem az egyetlen kezelőfelülete a fürtöknek. Ha valamilyen oknál fogva nem áll rendelkezésre a grafikus

felület, de a parancssoros igen, a node system32 könyvtárában megtalálható a cluster.exe program, amely a net parancshoz hasonlóan igen gazdag funkciókészletet nyújt, szinte mindent megtehetünk - ha éppen nem mindent. A szintaxisa roppant egyszerű:

```
cluster [cluster név] /option
```

vagy

```
cluster [cluster név] node [node név] /option
```

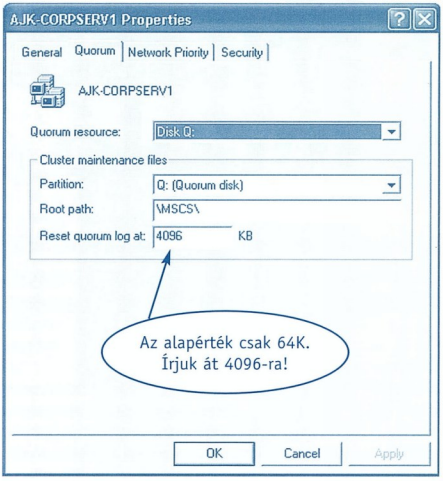
Így lehet például egy megosztáserőforrást létrehozni:

```
X:
MD TestShare
cluster . res "TestShare" /create
/group:"Group1" /type:"File Share"
cluster . res "TestShare" /priv
 path="X:\TestShare"
cluster . res "TestShare" /priv
 Sharename=TestShare
cluster . res "TestShare" /priv Remark="Teszt
 megosztas"
cluster . res "TestShare" /prop Description="Teszt
 megosztás"
cluster . res "TestShare" /priv
 security=Domain\User,grant,c:security
cluster . res "TestShare" /priv ShareSubDirs=1
cluster . res "TestShare" /AddDep:"Disk X:"
cluster . res "TestShare" /AddDep:"Network Name"
cluster . res "TestShare" /On
```

Nincs arra mód, hogy az opcióknak akár csak egy részét is ismertessük. A cluster.exe elsősorban akkor válik hasznossá, amikor egy rendszeresen ismétlődő feladatot kell végrehajtani, és ehhez szükséges a cluster állapotának lekérdezése, esetleg megváltoztatása.

A cluster beállításai

Mielőtt az egyes mappák tartalmában részletesen elmerülünk, érdemes megvizsgálni a cluster tulajdonságait. Kattintsunk a fa gyökerére a jobb oldali egérgombbal, majd a „Properties…" menüpontra. A telepítővarázsló kérdéseire adott válaszaink nagyobbik része itt található. A négy fülből igazából kettő izgalmas, a Quorum és a Network Priority.



A cluster tulajdonságai

Ha nem voltunk kellően megfontoltak, és új helyet kell találnunk a quorum adatbázisnak, azt itt tehetjük meg. Ezt azonban csak normál működés esetén lehet elvégezni. Egészen más a teendő, ha a quorum lemez megsérül, vagy az adatbázis megy tönkre. Lesz még hely és alkalom, hogy erről értekezzünk. A fülön van egy fontos paraméter, ez a quorum log mérete. Alapértelmezetten csupán 64K. A Q225081 leírása szerint ez kevés lehet, ha nagyszámú megosztásunk vagy egyéb erőforrásunk van, érdemes tehát megnövelni 4096K-ra. Ezt elég egyszer elvégezni és a clustert sem kell újraindítani a változások érvényre jutásához.

A „Network Priority" panel pontosan arra való, amire a neve utal. A varázsló kérdésére adott válasz itt módosítható. Akkor fordulhat ez elő, ha utólag telepítünk egy második kártyát. Ez a fül azonban csak a már illesztett kártyák sorrendjét változtatja meg, a szerepüket nem. Ha ez utóbbit is állítanunk kell, akkor a „Cluster configuration" konténer „Networks" mappájában kell megkeresnünk a hálózati csatolót, és módosítani a szerepét.

A teljesség kedvéért említsük meg a biztonsági lapot is. A fejlesztők nem vitték túlzásba a biztonsági beállítási lehetőségeket. Egy fióknak vagy nincs jogosultsága a cluster kezeléséhez, vagy mindent megtehet. Úgy érzem, az a szemlélet érhető itt tetten, miszerint a virtuális szerverek mégsem igazi szerverek *(még!)*. Nem önálló egységek, amelyek külön felelőst igényelhetnének, sokkal inkább a teljes fürt az, amely a felelősség határát meghúzza. Hogy ez a későbbiekben változni fog, az bizonyos.

A fent tárgyalt párbeszéd-paneleket ritkán fogjuk használni, hiszen nem naponta változik egy fürtözött konfiguráció. Annál inkább a napi munka része a csoportok és erőforrások kezelése, lássuk tehát, mi mindent tehetünk.

Csoportok

Ha az előre létrehozott közös lemezeket a varázsló kérdésére mind a cluster kezelésébe utaltuk, akkor már az első indításkor több csoportunk is van, amelyek nemes egyszerűséggel a „Disk Group 1..n" nevet viselik. Az első csoport kivétel csak, ő a „Cluster Group" Ez tartalmazza a cluster név, cluster IP cím és quorum erőforrásokat. Amikor munkahelyemen az első clustert üzembe helyezték, a szállítónk előre telepítette a clusterszolgál-

tatást az általunk megadott erőforrásokkal és virtuális kiszolgálókkal együtt. A csoportok neve megegyezett a virtuális szerverek nevével. Az első saját telepítésemig azt hittem, hogy a csoportnév és a virtuális szervernév egy és ugyanaz, de ez nem így van. Érhető is: vannak olyan csoportok, amelyek nem virtuális kiszolgálók, tehát nincs NetBIOS, csak csoportnevük. Az utólagos beállítás azonban hasznos, hiszen nem kell folyton a csoportnevek mellé rendelni a szerverneveket. Azt javaslom tehát, hogy minden csoportot, amely egyben virtuális kiszolgáló is, nevezzék át a rendszergazdák a virtuális szerver nevére, beleértve az első csoportot, vagyis a clustert is. Nem kötelező ez, csak egyszerűbbé teszi az életet. Az átnevezés nagyon egyszerű: jobbklikk a csoporton és „rename".

Új csoport felvétele hasonlóan könnyű: jobbklikk a fa gyökerén: new –> group. A létrehozott csoport nem tartalmaz erőforrásokat, persze tetszés szerint feltölthető újakkal vagy már meglévőkkel. Ez úgy végezhető el, hogy offline módba kell tenni az erőforrást, majd jobbklikk: „change group –> group name". Egy fontos dolgot azonban figyelembe kell venni, ez pedig a függőség.

A függőségek

A cikksorozat első részében már definiáltuk, hogy ez mit jelent, most az első erőforrások létrehozásánál a gyakorlatban is alkalmaznunk kell a korábban leírtakat. Nincs olyan KB cikk, amely minden egyes lehetséges erőforrást leír és meghatározza, hogy mely más erőforrásoktól függhet. Általánosságban a Q171791 foglalkozik a kérdéssel. Néhány ökölszabályt érdemes megfogalmazni, ezek könnyedén megvalósíthatók a gyakorlatban.

1. A lemezerőforrások soha ne függjenek más erőforrásoktól.
2. Nem függ más erőforrásoktól az IP cím és a Time service resource.
3. A „Network name" mindig az IP címtől függ.
4. A fájlmegosztás a „Network name"-től és a lemez erőforrástól függ.
5. A fájl- vagy adatbázis műveleteket végző erőforrások mindig függnek a fizikai lemeztől. (Pl.: WINS)
6. A hálózati szolgáltatások mindig függnek az IP címtől, néha a hálózati névtől is.

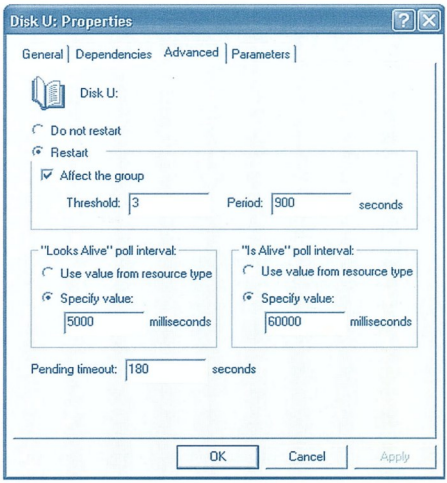
A szabályok nagyobbik része értelemszerű, magyarázatra sem szorul. Mindenképp érdemes egy kis időt szentelni a függőségekre, és akár kis téglalapokkal ábrázolni is lehet. Törekedni kell az egyszerű, átlátható ábrára. Úgy kell kialakítani a hierarchiát, hogy egy erőforrástól ne függjön kétféleképp egy másik erőforrás. Például ha a fájlmegosztás már függ a hálózati névtől, akkor nem kell függővé tenni az IP címtől, hiszen már a hálózati néven keresztül amúgy is függ tőle. Ez nem minden erőforrás esetén tartható. A WINS és DHCP erőforrások például követelik mind a hálózati név, mind az IP cím függőséget, nem is lehet definiálni őket másképp. A függőség a gyakorlatban úgy érvényesül, hogy egy erőforrás indítása csak akkor kezdődik el, ha már minden erőforrás, amelytől függ, sikeresen elindult. Minél több tehát a függőség, és minél hosszabb a függőségi sor, annál később indulnak el a sor végén álló erőforrások. A jelenlegi ismeretek alapján nem meglepő már, hogy az erőforrások függőségét csak azonos csoporton belül lehet létrehozni. A különböző csoportok ugyanis elkerülhetnek az azonos állomásról, így a függőség érvényesítését nem lehetne garantálni. Végezetül ajánlom, hogy a triviális függőségek mellett tárjuk fel a logikai függőségeket is. Például egy clusterre fel nem készített, de általános szolgáltatás-erőforrásként definiálható vezetői információs rendszer függhet az SQL Servertől, amely az adatokat szolgáltatja neki, jóllehet ezt a függőséget semmi nem követeli meg, csak a rendszergazda ítélőképessége.

Át- és visszaköltözés, újraindulás

Ha már a függőségeket megemésztettük, ismerkedjünk meg az utolsó fontos fogalmakkal a gyakorlatban: az át- és visszaköltözéssel, valamint az újraindulással. Adódik az eset: egy erőforrás abnormális állapotba kerül. Mit tegyen ekkor a cluster? Egyáltalán hogyan értesül erről? Elég fontos kérdések ezek, tulajdonképpen a rendszer szíve-gyökere ez a funkció. Mi másért tartanánk egy clustert, ha nem azért, hogy éppen ezt tudja.

A cluster szerviz egyik igen fontos feladata az erőforrások állapotának figyelése, valamint annak ellenőrzése, hogy az erőforrás valóban ellátja-e a feladatát. A kétféle ellenőrzést „Looks alive"-nak *(„úgy tűnik, él")* és „Is alive"-nak *(„valóban él")* nevezhetnénk el az erőforrás tulajdonságlapja alapján. Az első esetben az ellenőrzés abból áll, hogy a szolgáltatás megnézi a regisztrációs adatbázis cluster részét, vajon minden rendben van-e. Kiindulásnak nem rossz ez az ellenőrzés, mert az erőforrást nem terheli, elég egyszerű és gyors, csak az a bökkenője, hogy mindig a saját korábbi vizsgálatára támaszkodik, tehát nem igazi ellenőrzés. Ezzel a vizsgálattal valóban csak azt lehet mondani: „Úgy tűnik, hogy működik." Ez az ellenőrzés sokáig nem folytatható, ezért néha meg kell vizsgálni azt a futó programszálat *(program thread)*, amely az erőforrás maga. Ez az „is alive" vizsgálat. Ez már igazi ellenőrzés, cserébe egy icipicit foglalja és fogja a rendszert. Ez már csak így szokott lenni.

Nos, tegyük fel, hogy egy ilyen „is alive" vizsgálat során azt tapasztalja a cluster szolgáltatás, hogy nem válaszol a programszál. Ekkor válik igazán fontossá, hogy az erőforrás és a csoport, amelyben az erőforrás székel, milyen átköltözési szabályokkal rendelkezik.



Átköltözési szabályok egy erőforrásnál

A panel alapján akár azt is megtehetjük, hogy hiba esetén nem teszünk, illetve a rendszerrel nem végeztetünk semmit. *(„Do not restart")*, bár még nem sikerült igazán olyan életszerű szituációval találkoznom, ahol ez lett volna a helyes választás. Szinte bizonyos, hogy legalábbis újraindítjuk az erőforrásunkat. Meg kell adnunk viszont, hogy az erőforrás állapota hatással legyen-e a csoport állapotára *(„Affect group")*. Ha azt választjuk, hogy nem, akkor a helyreállítási procedúra a következőkből áll:

1. Ha még nem fagytak le, akkor a clusterszolgáltatás leállítja a hibás erőforrástól közvetve vagy közvetlenül függő erőforrásokat.
2. A „Period" *(„időköz")* mezőben megadott másodpercen belül maximum a „threshold" *(„próbaküszöb")* mezőben meghatározott alkalommal megpróbálja újraindítani az erőforrást.
3. Ha sikerül az újraindulás, akkor a függő erőforrásokat is megpróbálja elindítani.
4. Ha nem sikerült a start, akkor az erőforrás és minden tőle függő más erőforrás is „Failed" *(„Hibás")* állapotban marad.
5. Megvizsgálja, hogy a függő erőforrások hibás állapota érinti-e a csoportot. Ha nem, akkor a helyreállítási procedúra leáll.

A legfőbb fegyvert, az átköltözést ilyenkor nem használjuk. A költözés ugyanis a csoport joga, mi viszont nem kívántuk eszkalálni a problémát csoportszintig. Jó ez így? Hiszen nem tudhatjuk, hogy mi a hiba oka, s talán „máskor, más helyen" *(értsd: a másik állomáson)* működhetne a szolgáltatásunk. Bármilyen fura, lehetséges, hogy ez jó megoldás. Tegyük fel, hogy több száz megosztást definiáltunk clustererőforrásként. Ha egy könyvtárat véletlenül törlünk, a hozzá tartozó erőforráscsoportnak sokáig tartana átköltözni a másik állomásra, ráadásul felesleges lenne az átköltözés, hiszen a többi erőforrásnál ez csak kiesés, míg a törölt könyvtárnak úgyis mindegy, ha egyszer törölni akartuk. Fontoljuk meg tehát, hogy érintse-e az erőforrás összeomlása a csoportot. Ez néha szükséges, néha nem – egyértelmű tanácsot csak konkrét környezetben lehet adni.

Bevallom, hogy az előbb kicsit csaltam. Hiszen az 5. pontban az is előfordulhatott volna, hogy a függő erőforrás már érinti a csoport átköltözését. Igen, ekkor valóban megkezdődik az átköltözés. Ha ezt szeretnénk elkerülni, akkor át kell nézni minden, az erőforrástól függő más erőforrást és a megfelelő jelölőnégyzetben eltüntetni a kis pipát.

Ha az erőforrás magával rántja a csoportját, akkor megindul az átköltözés a... hova is pontosan? Nos, az erőforrás által kijelölt másik állomásra. *(Tulajdonságok panel, „Possible owner" bejegyzés)* Advanced Server esetén ez kissé nagyzolásnak tűnik, hisz már csak egy szerverünk maradt, ahová a csoport átgördülhet, Datacenter Servernél viszont három vagy négy állomás is alkothat clustert, turkálni lehet a választékban. Ám ennek az ellenkezője is előfordulhat, hogy nincs is másik szerverünk. Miért? Lehetséges, hogy nincs is második állomás? Igen, lehet. A cluster állhat akár egy node-ból is, ekkor nem lehetséges az átköltözés. Akkor fordul ilyesmi elő, ha a cluster első állomását előbb állítjuk üzembe, mint a másodikat, de gondolva a jövőre, már az első node-on úgy állítunk be mindent, mintha teljes kiszolgálófürtünk lenne. Az új gép csatlakozásakor a cluster szerencsére elvégzi nekünk, hogy minden erőforrásunknak legyen lehetséges tulajdonosa az új állomás is, ennek ellenére fontos a következő szabály: egy csoportban minden erőforrásnak legyenek ugyanazok a node-ok, ugyanolyan sorrendben preferált szerverei! Ez nem zárja ki, hogy egy csoportnak továbbra is csak egy preferált állomása legyen. Lehet, hogy egy szoftvert *(pl. Exchange)* ugyan clusterre telepítettük, clustererőforrásokként működnek a szolgáltatásai, de nem rendelkezünk egy második kiszolgálólicenccel ahhoz, hogy a második node-ra is felrakhassuk szegényt. Ekkor bizony be kell állítanunk, hogy minden Exchange erőforrásnak, sőt az egész csoportnak csak egyetlen állomás legyen a birtokosa. Hiba esetén újraindítás lehetséges, átköltözés nem.

Mi történik akkor, ha a csoport nem tud elindulni azon az állomáson, amelyre át szándékozott költözni és megpróbál azonnal visszaköltözni az első állomásra? Az örökciklust azért kivédték a tervezők. A csoport tulajdonságai között is megadhatók átköltözési szabályok. Jobbklikk: „A csoport neve" –> Properties –> Failover.

A threshold érték szabályozza, hogy hányszor indulhat újra másik állomáson egy csoport egy átköltözési periódusban. Ezután a cluster szerviz offline állapotba teszi a csoportot. A periódus hosszát a Period érték adja meg. Vagyis: ha 11-szer indulna újra a csoport, de a küszöbérték 10, akkor a cluster kikapcsolja a csoportot, amíg le nem jár az átköltözési periódus. A cluster szerviz újraindításával új periódust lehet kezdeni.

Átköltözési szabályok egy csoportnál

És a visszaköltözés? Alapvetően két lehetőségünk van: a kiszolgálókra bízzuk, vagy magunk végezzük el. Ha magunk végezzük el, az biztonságos és a megfelelő időben történik, csak kényelmetlen. Egyrészt kézzel kell megoldani, fejben kell tartani a feladatot, másrészt a visszaköltöztetést szinte bizonyosan egy kevésbé frekventált időben, este vagy éjszaka kell végrehajtani, ezt pedig senki sem szereti. Elvégzi a visszaköltöztetést a cluster is, méghozzá pontosan úgy, ahogy azt mi szeretnénk. Választhatjuk azt, hogy a csoport regenerálódás után azonnal röppenjen vissza a helyére, de megadhatjuk azt is, hogy ez a mozgás egy adott napszakban történjék meg. Mérlegelni itt azt kell, hogy az üzemeltetési paraméterek *(pl. válaszidő)* elsőbbséget élvez-e a rövid kieséssel szemben, amelyet a napközbeni visszaállás okoz.

Most már minden tudásunk megvan, hogy erőforráscsoportokat és erőforrásokat hozzunk létre. A következő alkalommal a fájl- és nyomtatóerőforrásokkal kezdünk.

Lepenye Tamás, MCSE 2000
lepenyet@imal.hu

Ki mivel ♥?

Farkasokkal Netmonozó

Emlékeznek még a ... típusú nyomtató esetére, mely beállt a fürtbe? Azt a cikket Lepenye Tamás követte el, a NetMon sorozatba „furakodva". Most pedig én fogok az ő szakterületébe belekotyogni egy fürtküzdelem kapcsán.

Adva van egy cég, korlátlan pénzügyi lehetőségekkel. Igen, egy bank. Megveszik a világ egyik legdrágább clustervasát, mivel nekik X kilences rendelkezésre állás kell. Megy is a fürt szépen - egy darabig. Egyszer csak telefonhívásra lesznek figyelmesek a rendszergazdák: a mission critical alkalmazás nem elérhető a fürtön. Mi a hibaüzenet? A szolgáltatás megtagadva, mert a **tartományvezérlő** nem válaszol az autentikációs kérésekre. Hmmm. Létrejött egy Achilles sarok a fürtön kívül? A single-point-of-falilure *(SPOF)*, melyre nem gondoltunk? Néhány gyors ellenteszt *(kijelentkezés, bejelentkezés, MSSQLServer stop-start stb.)* alapján kiderítik, hogy a tartományvezérlők igenis válaszolnak. SPOF-helyzet amúgy sem lehet, hisz nem egy darab tartományvezérlő van, hanem öt. Mire idáig jutnak a nyomozásban, a hiba magától megszűnik, az alkalmazás szépen fogadja a munkaállomások kéréseit. Múló rosszullét csupán?

A második felvonásra több napot várni kell. De eljön. A legádázabb munkaidő kellős közepén a fürtalkalmazás ismét kirugdalja a józereket. A hiba oka ugyanaz: „No DC has answered for the authentication request." Számítva arra a lehetőségre, hogy a hiba esetleg ismét hamar elmúlik, az előző teszteket nem hajtják végre *(azaz nem vacakolnak a DC-k válaszainak tesztelésével, mert úgyis tudják, hogy válaszolnak, hisz a fürtön kívül senki nem panaszkodik)*, hanem futás a clusterhez, NetMon indít. Mit látnak? A mission critical app. folyamatosan kapja a bejelentkezési kéréseket, és azonmód továbbítja az egyik DC-nek. A NetMon trace tartalma:

- WINS lekérdezés, 1C *(tartományvezérlő)* rekordokért
- WINS válasz, benne mind az öt DC
- Netlogon SAM Logon Request mind az ötnek
- amire nem jön válasz egyiktől sem

Ejnye no, mi a csuda van itt? Passz. Elmentik a trace fájlt, hátha jól jön még. Megpróbálják ugyanezt megnézni a DC oldaláról. Futás, futás, mert megjavul! És valóban, mire odaérnek, a fürt kikeveredik a slamasztikából, és minden megy tovább. Ha már a NetMon kicsúszott a kezük közül, legalább körülnéznek az ominózus DC-n. Az Event Viewer tanúsága szerint semmi nem történt. Nincs mit tenni, várniuk kell a következő botlásig. Nagyon ciki az ügy, a tízmilliós fürt botladozik. Hol háromóránként, hol kétnaponta, hol egy percre, hol ötre kiakad. Mi az, hogy az X kilences méregdrága rendszer megbízhatatlanabb, mint a sarki fűszeres PC-je? Munkakönyvszag terjeng a levegőben. Elhatározás születik külső szakértő bevonásáról.

A vadászat

A Nagy Hibavadászatra ezúttal meghívtak engem is. Első dolgom az Event Logok áttanulmányozása. Semmi különös nincs bennük, csupa info ikon. A fürt köszöni jól van, a heartbeat ismét dobog stb. Két nap üldögélés után a fürt hál'Istennek megint megborul. Nem is baj, mert olyan eszméletlen hideg volt a szerverszobában, hogy a fogunk csattogása elnyomta a gépek búgását. Végre egy kis mozgás, lehet rohanni. Nosza, futás a DC-khez, NetMont elő, hálózati forgalmat elkapni! És lett nagy csodálkozás. Mert az összes DC válaszol, de valami miatt nem kapja meg a fürt! Pedig tisztán látszik: ugyarra a MAC Addressre megy a válasz, ahonnan a kérdés jött. Hova tűnik? Talán rossz a fürt egyik csomópontjának kártyája? Ennek ellentmond, hogy minden más hálózati forgalmat megkap. Sőt, a DC nézőpontjából figyelve az eseményeket, a Netlogon SAM Logon Response-t is megkapja. Hmmm.

Agyak őrült zakatolása hallik, míg végre valaki megszólal:

„Az nem a fürt, hanem a router MAC Addresse!"

Lám, mi mindenre jó egy olyan kolléga, aki fejből tudja a vállalat összes gépének MAC Addressét! Tényleg! Hisz a DC meg a fürt két külön alhálózaton van! A router a bűnös, eldobálja a válaszcsomagokat. De hát miért? Mivel ezt nehéz lenne tőle megkérdezni, faggasuk tovább az imént elkapott hálózati forgalmat, hátha ki lehet sütni belőle valamit! Kettőt kattintok az egyik csomagon. Első ránézésre semmi különös. De ismét bekiabál az iménti kolléga:

„Az nem a fürt IP címe! Az valami szemét!"

Na igen. Ennyit tesz a helyismeret. Jókora lépéssel közelebb kerültünk a probléma gyökeréhez: úgy tűnik, mintha a Logon Requestet nem is a fürt küldené, ergo a válasz sem hozzá megy vissza. Hamisított IP cím egy banki hálózaton? Hackert fogtunk? De jó lenne! Lehetne hívni a sajtót! Hírnév, siker, pénz, csillogás! Most kisebb nyomozás következik, hogy megállapítsuk, honnan jön a szemét. Most már sehonnan, mert közben ismét meggyógyult a hálózat. A szemét IP cím elvileg senkié, ilyet a DHCP Server sem ad ki. Azonban beletekintve a legelőször lementett fájlba *(mely a fürtön készült)* kiderül, hogy a Netlogon SAM Logon eleve szemét IP címmel kerül ki a netre. Ha figyelmesebbek lettek volna, majdnem egy hetet megspórolhattak volna! Innen már csak néhány kattintás, és megvan a döbbenetes eredmény: a szemét nem más, mint a fürttagok magánhálózatán *(heartbeat network)*, a **crossover kábelen** használt IP cím, amit teljesen jogosan állított be a telepítőszemélyzet akármire, mert az a cím soha nem kerül ki a publikus hálózatra. Kivéve, ha mégis. Egy-két kattintás, és kiderül a „hiba" oka: valaki átállította a heartbeat hálózatot úgy, hogy „both" legyen, azaz mind heartbeat, mind ügyfélforgalom céljára igénybe vegye a fürt. Jó kérdés, hogy miért zavarodott bele a node, és adott fel az egyik kártyán olyan csomagokat, amelyek a másik kártya IP címét viselték, de talán még izgalmasabb az, hogy ki a fene állította át? Ki volt az a pancser? Hát ezt majd a banki dolgozók lerendezik maguk közt, én visszaállítom normálisan private hálónak, veszem a kalapom és

megyek. Pár nap alatt kiderül, hogy egyedül két embernek volt joga ilyesmihez, de mind a kettő tagad. Mit bánom én, fő a sikerdíj!

A negyedik felvonás körülbelül egy hét múlva következett, amikor már senki nem számított rá. A heartbeat hálózatot valaki megint átállította publikusra. Ezalkalommal gyorsan cselekedtek, és visszaállították a heartbeatet privátra. Hurrá. Három hét telik el békében. A fürt dolgozik. Egyszercsak megint közlik, hogy átállította valaki. De ki tette? Mégiscsak hacker van? A két jogosult személy tagad, sőt alibit is felmutat: azon a héten végig tanfolyamon voltak. Tekintettel a „bűntény" tökéletesen értelmetlen voltára gyanakodni kezdtem, hogy talán magától áll át időnként. Vajon miért?

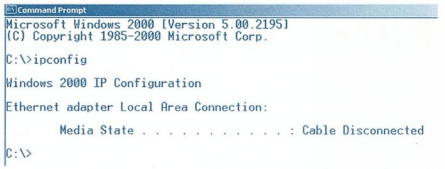
Dr. Watson a helyszínen összeszedi a legkisebb nyomokat is. Csikkek, lábnyomok, Event Logok kerülnek a nylonzacskókba. Lássuk, hátha valami mégiscsak megbújik itt? A fürt logjában mint korábban említettem, csupa semmi áll. A heartbeat **ismét** a privát hálózaton fut. Mi az, hogy **ismét**? Gyűjtsük csak ki ezeket az eseményeket! Három ilyen van, ebből az utolsó kettő - pont a behülyülés napjára esik. Vajon a harmadik is? Elküldtem ímélben a három dátumot a bankosoknak, mondják meg mi jut róluk az eszükbe. Jön a válasz: ezeken a napokon állítódott át a fürt. Nyomon vagyunk! Hopp, mégegy válaszlevél beesik, az utolsó utáni, képesítés nelküli rendszergazdától: ezeken a napokon szerelték a klímát. Nocsak-nocsak! Mégis egy Achilles sarok, egy SPOF áll a háttérben? De mi köze lehet a klímához? A fürt jól bírta a klimatizáló berendezés hiányát, de a kijavítást nem! Klimaszerelés=fürtbehülyülés. Vajon miért?

A probléma első része: hardverhiba

Nem csigázom tovább Önöket, elárulom a titkot. A heartbeat hálózat egyszerű, házi berhelésű keresztkábellel *(crossover)* volt kialakítva. Íme a SPOF. A kábel ugyanis kontakthibás volt, ami mindig akkor jelentkezett, amikor valaki elsétált a cluster MÖGÖTT! Ki járkál a gépek mögött? Még a takarítószemélyzet sem. De bizony a klímaszerelők igen! Megbökik a kábelt, a heartbeat elhal, a fürt pedig – okosan – átáll a publikus hálózat használatára. Amint a kábel visszatér eredeti helyzetébe, az Eventlogban megjelenik a bejegyzés: a heartbeat **ismét** a privát hálózaton fut. Már csak az a kérdés, hogy ennek miköze a heartbeat IP címek külső hálózaton észlelt felbukkanásához?

A probléma második része: Media Sense

A Windows NT 4.0 Enterprise Edition fürtje óriási hiányossággal rendelkezett: nem vette észre, ha a publikus hálózati kapcsolat veszett el. Egyszerűen nem következett be ilyen esetekben failover, mert a fürt jól van. Ennek a hiányosságnak az az oka, hogy az NT4 maga képtelen detektálni a hálózati kapcsolat megszakadását. Nem úgy a Windows 2000! Ha kitépjük a kábelt az Ethernet kártyából, azonnal jelzi, hogy oda a kapcsolat. Erre a fícsörre a fürt is épít, és ha a felhasználói hálózattal megszakad a kapcsolat, költözik a hivatal. Ám a Media Sense eléggé furcsán működik. Megpróbálta már valamelyikük megállapítani a gép IP címét, ha nincs bedugva a hálókábel? NT4 esetén ilyenkor is volt IP cím. A Windows 2000 azonban ezt mondja:



Windows 2000: Ha nincs hálózat, akkor nincs IP cím sem!

Nem is lenne ezzel semi baj, ha csak az IP cím tűnne el. De sajnos kiszalad a memóriából az egész perepurty, kártyameghajtóstól, mindenestül, méghozzá mindkét Node-on. On-demand Plug-and-Play. Úgy eltűnik, mintha sosem lett volna. Eltűnik a Cluster fennhatósága alól is.

S amikor visszadugjuk a kábelt? **Új kártya születik!** Ráadásul mindkét Node-on. Nincs mihez párosítani. Ha csak az egyik Node-on tűnt volna el, az IP cím alapján visszakerült volna a helyére. De a crossover kábel miatt mindkét Node elveszítette. Na ez a baj. Menet közben beszületik a fürt alá egy új kártya. Vajon mit tegyen a fürt? Publikus, vagy heartbeat hálónak vegye fel? Tanácstalanságában both, azaz mindkét szerepre állítja be. S ezzel a történetnek majdnem vége. Hátra van még a végső megoldás.

A Megoldás

A megoldás egy Knowledge Base cikkben bujkál *(Q254651)*. Crossover kábel használata esetén tiltsuk le a Media Sense „szolgáltatást". Vagy ne használjunk házi kábelt. Vagy ne crossovert. A figyelmes olvasók talán észrevették: ezt a beállítást Lepenye Tamás a decemberi Farkasokkal már megírta. Neki van igaza. Aki fürtöt használ, legyen tudatában annak, mit tesz. Legyen óberokos, olvassa át az összes vonatkozó információt. Az összes Knowledge Base cikk szabad szövegesen keresheto formában megtalálható az **[1]** címen.

Szívműtéthez *(heartbeat)* ne a csontkovács szerszámait használjuk. Ennyit erről.

Fóti Marcell
marcellf@netacademia.net

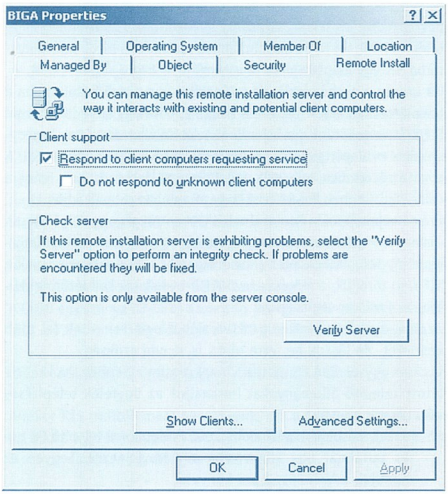
A cikkben szereplő URL-ek:
[1] http://support.microsoft.com

RIS
(III. rész)

A mai alkalommal a RIS kiszolgáló beállításait nézzük végig. A RIS összes felügyeleti szerve be van építve az Active Directory Users and Computers MMC snap-inbe, így ezzel a megszokott eszközzel *(szinte)* minden RIS-karbantartási feladatot elvégezhetünk.

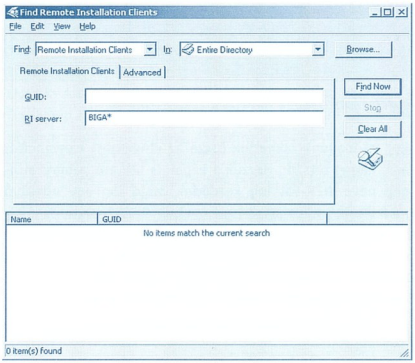
A RIS kiszolgáló beállításai

Amikor a RIS kiszolgálóobjektumának tulajdonságait előhúzzuk *(AD Users and Computers konzolon a kiszolgáló objektumán jobb klikk - properties)*, ott a következő kép fogad minket.



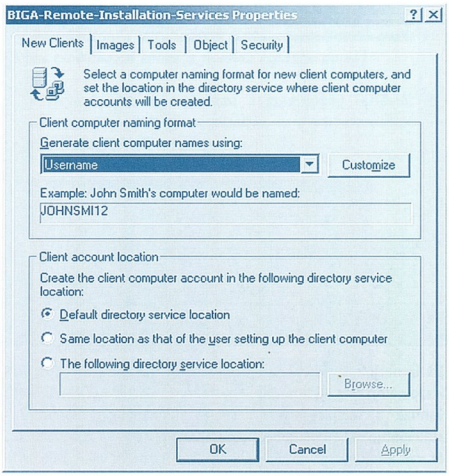
Remote Install tulajdonságok

Client support – itt lehet be és kikapcsolni a RIS kiszolgálást. A fenti képen látható beállítás minden ügyfélnek válaszolni fog, aki hozzá eljut, míg ha bekapcsoljuk a második jelölőnégyzet is, akkor csak azoknak fog válaszolni, akiknek van már megfelelően előkészített számítógépazonosítója a tartományban. Értelemszerűen, ha egyik pipa sincs a helyén, akkor a RIS kiszolgálót kikapcsoltuk.
Verify Server – ellenőrizhetjük vele, hogy minden rendben van-e a kiszolgálóval – nem tesz mást, mint a risetup.exe-t indítja –check kapcsolóval, és csak akkor elérhető ez, ha az AD Users and Computers snap-int magán a kiszolgálón futtatjuk.
Show Clients – megmutatja azokat az ügyfeleket, akiket az adott RIS kiszolgálóról telepítettünk. Egyszerű keresés az Active Directory-ban.



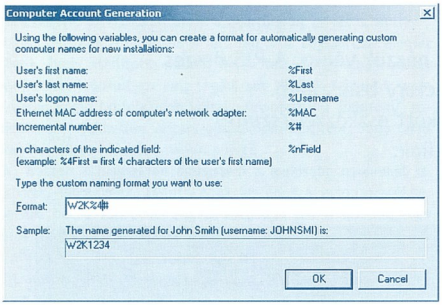
RIS ügyfelek felkutatása a tartományban

Advanced Settings – itt tulajdonképpen az ügyfélszámítógépek elnevezését szabályozhatjuk, valamint a telepítőkészletek hozzáadása/eltávolítása is innen lehetséges.



Az ügyfélszámítógépek

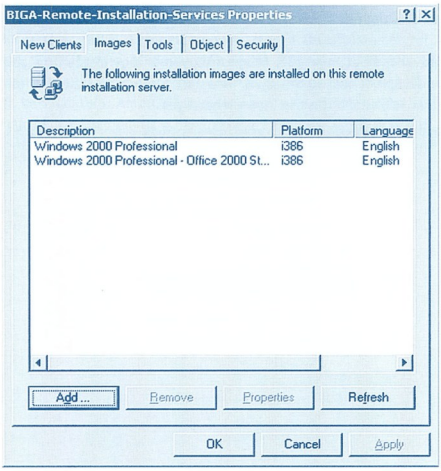
Nézzük meg közelebbről először az ügyfelek elnevezéseit. Itt tudjuk megmondani, hogy mi is legyen az Automatikus telepítéskor létrehozott gépnév az Active Directory-ban. Választhatjuk a telepítést végző felhasználó nevét, vagy a vezetéknév és a keresztnév kombinációit, vagy NP plusz a gép MAC címe, de választhatunk egyedi elnevezést is.



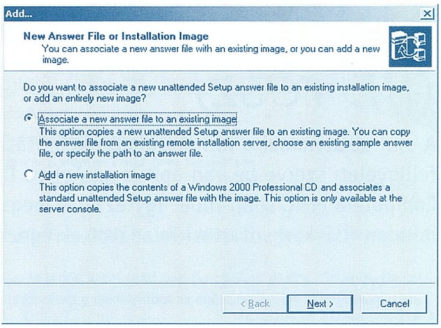
Ügyfélszámítógépek neveinek testreszabása

Ilyenkor mi magunk rakhatjuk össze az elnevezési konvenciót, majdnem ahogy tetszik. Nézzük a képen is látható példát: minden gép neve W2K-val fog kezdődni, majd jön egy növekvő szám, a mi esetünkben max. 4 számjegyű lehet. Itt mindig a következő legkisebb még használaton kívüli számot fogja felajánlani az Ügyféltelepítő Varázsló. Ezen a néven fogja létrehozni a gépazonosítót a tartományban. Ha nem adunk meg számot, akkor alapértelmezés szerint 2 számjegy kerülhet a végére.

Izgalmasabb az Images fül itt a beállítások közt.



Itt láthatjuk az összes telepítőkészletet, amelyek az adott RIS kiszolgálón megtalálhatóak. Egy telepítőkészlet két részből áll: vannak a telepítéshez szükséges állományok a RemoteInstall\Setup\%Language%\Images\<image dir>\ könyvtárban, és van a felügyelet nélküli telepítéshez használt válaszfájl – SIF fájl. Több SIF fájlt is rendelhetünk ugyanazokhoz a telepítendő állományokhoz, de egy SIF fájl mindenképp szükséges egy garnitúra telepítőkészlethez. Ebben a menüben adhatunk hozzá válaszfájlokat *(Add gomb)* a meglevő állományokhoz.



Telepítőkészlet varázsló

Ha egy teljesen új telepítőkészletet akarunk létrehozni, *(mondjuk több nyelvi verzióra is szükségünk van)*, akkor válasszuk az Add a new installation Image varázslót. Ilyenkor bekéri a Windows 2000 telepítő CD-t, fel fogja másolni annak tartalmát a megadott könyvtárba, és egy alapértelmezett válaszfájlt is hozzá fog rendelni.
Ha azonos a nyelvi verzió, de szükségünk van eltérő beállításokra a telepítéseknél, akkor nincs más dolgunk, mint egy előre létrehozott SIF fájlt összekötünk a már fentlevő telepítő állományokkal. Legegyszerűbb az alapértelmezés szerint bepakolt SIF fájlt szerkeszteni. A következő részben lesz róla szó hogy mit lehet és kell tenni a válaszfájllal annak érdekében, hogy az valóban jól működjön.
Ezek a válaszfájlok a RemoteInstall\Setup\%Language%\Images\<imagedir>\i386\templates könyvtárban tárolódnak. Amikor letörlünk egy telepítőkészletet *(Remove gomb)* tulajdonképpen csak a SIF fájlt töröljük, a mögötte levő X Mb-ot nekünk kell letörölni külön, az Explorer segítségével.
Azonos nyelvi és service pack verzióból nem tehetünk fel több készletet, de lássuk be, erre nincs is semmi szükség.
Viszont egyszerűen elérhetjük, hogy rögtön a Service Packkal ellátott telepítő állományokat használjuk az ügyfelek telepítésénél. Lássuk hogyan:

- Fel kell másolnunk a Windows 2000 Professional telepítő CD teljes tartalmát egy könyvtárba a merevlemezre. Legyen ez mondjuk d:\win2ksp2.pro könyvtár.
- Ezek után fogjuk a Service Packot, azt is bontsuk ki /x kapcsolóval.
- Majd az i386\Update könyvtárából futtassuk a következőt: update –s:<dir> – annak könyvtárnak a nevét és teljes elérési útját kell megadni, ahová a CD-t felmásoltuk, nem benne az I386 könyvtárat! Tehát az előbbi példánál maradva a parancs így fog kinézni: update –s:d:\win2ksp2.pro.
- Mindezek után hozzáadhatjuk az így előkészített telepítőállományokat az Add new installation image varázslóval, vagy a risetup.exe elindításával.
- Utolsó lépésként az eddig is használt válaszfájlt hozzárendeljük a telepítőállományokhoz, és máris kész van a friss, ropogós telepítőkészlet.

Az ügyfélgépek előkészítése

Biztonsági szempontból mindenképp megfontolandó, hogy a RIS-es ügyfélgépeknek jó előre létrehozunk azonosítót a tartományban. Ezt hívjuk pre-stagingnek. Miért is jó ez? Nos, a felhasználó még véletlenül sem fogja tudni elindítani a telepítést,

illetve nem kell a felhasználóknak extra jogokat osztani az Active Directoryban ahhoz, hogy telepíteni tudják a saját gépeiket, ha a rendszergazda előkészíti a gépaccountot is.
A cikk legelején már láttuk, hogy a RIS kiszolgáló gép accountjának Remote Install tulajdonságai közt találhatjuk mindjárt azt a beállítást, hogy a RIS kiszolgáló mely ügyfeleknek válaszoljon. A lenti kép beállítása azt mutatja, hogy a RIS kiszolgáló be van kapcsolva, tehát válaszolni fog az ügyfelek kéréseire válogatás nélkül. Ha a második négyzetben is ott figyelne a pipa, akkor csak azoknak az ügyfeleknek válaszolna, a melyeket már előkészítettünk. Ezt két okból lehet hasznos bekapcsolni. Ha a hálózaton van más olyan ügyfél, amely PXE-t használ, de nem RIS-sel akarunk rá telepíteni dolgokat, akkor ezeknek az ügyfeleknek a RIS kiszolgáló nem fog válaszolni. Másik ok pedig, hogy senki nem fog tudni kalóz módon felhúzni egy gépet RIS-sel, ha az nincs előkészítve az Active Directoryban, ugyanakkor mi sem fogunk tudni felvenni újakat csak akkor, ha valamilyen módon tudjuk a gép GUID-ját, és nem kell hozzá elindítani az ügyféltelepítő varázslót.

A RIS kiszolgáló ki/be kapcsolása

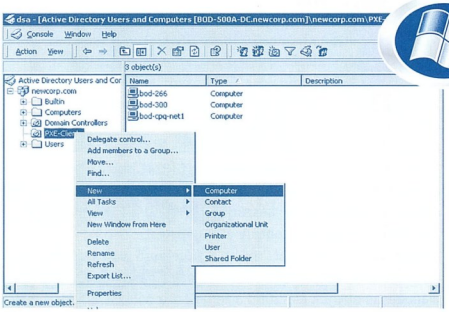
Ahhoz, hogy accountot tudjunk létrehozni az ügyfélgépnek a RIS telepítéshez, tudnunk kell annak egyedi azonosítóját, a GUID-ot. Ezt BIOS-ból meg lehet tudni, vagy ha elindítjuk az ügyféltelepítő varázslót az F12-vel, akkor miután beléptünk, ki fogja irni a gép nevét *(amit generált neki a kiszolgáló)*, a GUID-ot, és a RIS kiszolgáló nevét. Ezen a ponton le kell kapcsolni az ügyfélgépet, elég megjegyezni a gép nevét, majd ezt az ügyfélaccount-ot lehet manipulálni annak érdekében, hogy a megfelelő névhez a megfelelő GUID tartozzon.
A GUID egy csúnya 32-bites azonosító, amely a hálózati kártyához kötött. A PXE-t ismerő ügyfeleknél a GUID általában valahogy így néz ki:

{1f988641-3046-780a-1236-105146887113}

Amikor RIS indítólemezt használunk, ott a GUID nem más, mint az adott ügyfél hálózati kártyájának címe *(MAC címe, rengeteg 0-val az elején, hogy kijöjjön a 32 bit)*:

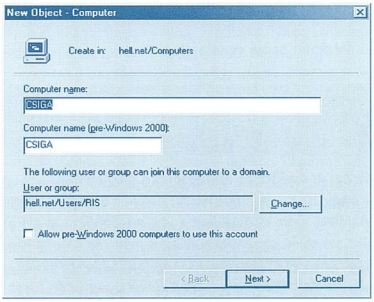
{00000000-0000-0000-0000-0050568F0102}

Miután megtudtuk a GUID-ot, az ügyfél előkészítéséhez az Active Directory Users and Computers-t használjuk.



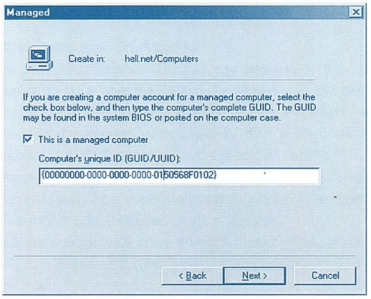
Új azonosító létrehozása

Ha a „Create a new computer" *(új számítógép létrehozása)* opciót választjuk, megjelenik a következő párbeszédablak és beírhatjuk a számítógép nevét, valamint megadhatjuk azt, hogy az ügyfelet ténylegesen ki léptetheti be a tartományba. Alapbeállításként ez Domain Admins, ha ezt így hagyjuk, akkor hiába készítjük elő az ügyfélgép accountot, a RIS telepítés nem fog tudni elkezdődni csak olyan account segítségével, aki ennek a csoportnak a tagja. Tehát ide azt a csoportot kell megadni, amelynek tagja az, aki telepíteni fogja a gépet, vagy megadhatunk persze egy felhasználót is.



Új számítógép létrehozása. Figyelmesen állítsuk be, ki lesz képes csatlakozni ehhez a fiókhoz (change gomb)!

A következő párbeszédablakban a számítógép GUID-jét kell megadnunk *(ha a RIS nincs telepítve, ez a párbeszédablak nem is jelenik meg)*.



GUID

Ezt a párbeszédablakot csak RIS kiszolgálókon vagy olyan kiszolgálókon és ügyfeleken tudjuk használni, ahol előzőleg telepítve lett az adminpak.msi.

A RIS kiszolgáló a GUID segítségével ellenőrzi, hogy az ügyfél létezik-e már az Active Directory-ban.

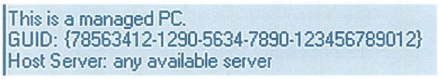
Egy GUID egy ügyfélhez tartozhat, ezért ha az ügyfélgépből úgy nyerjük ki a GUID-ot, hogy elindítottuk már egyszer a telepítőt, akkor először onnan kimásolhatjuk (CTRL+C) a GUID-ot, azt az accountot töröljük az AD-ból, majd létrehozzuk egy nekünk megfelelő néven, és itt beszúrjuk (CTRL+V) a kimásolt GUID-ot. Így biztos, hogy nem rontjuk el a gépelést.

Ha megadtuk a GUID-ot, akkor a következő párbeszédablakban megadhatjuk, hogy melyik RIS kiszolgálót használja az ügyfél. Ha manuálisan akarjuk elosztani a terhelést a hálózat RIS kiszolgálói között, hogy az egyes kiszolgálók csak a saját ügyfeleik kérését fogadják el, akkor itt pontosan megadhatjuk a RIS kiszolgáló nevét, amelyhez az adott ügyfélnek fordulni kell. Azonban ha ez a kiszolgáló nem működik, akkor más sem fogja az ügyfeleket kiszolgálni hiába elérhető még más RIS kiszolgáló is a környéken.

Ezután jön egy összegző képernyő, és már csak egy Next, és elő is van készítve az ügyfélgép azonosítója.

De nézzük csak meg ezeket a GUID-okat! Ha a BIOS-ból kiolvasott GUID-ot gépeljük be, akkor az így néz ki:

Az összegző ablakban pedig ez jelenik meg:



Nem egyezik azzal, ami itt az összegzésben megjelenik!

Közelebbről megvizsgálva észrevehetjük, hogy az összegző képernyőn megjelenített szám első 16 bájtja van megfordítva, a maradék 16 pedig a helyes sorrendben jelenik meg. Az információ, amit begépeltünk *(12345678901234567890123456789012)*, „wire" formátumú *(a hálózaton keresztül ebben a formában továbbítódik)*. Amikor a GUID megjelenik, „pretty print" formátumúvá alakul, ami a következőképpen néz ki:

{78563412-1290-5634-7890-123456789012}

A GUID-ot mindkét formában beírhatjuk, a program wire formátumúra alakítja és így menti az ügyfélgép account információi közt. A felhasználói felületen a GUID-t mindig „pretty print" formátumban jelenik meg.

A szükséges jogosultságok

Ha a RIS-es telepítést maga a felhasználó fogja a saját felhasználói nevével és jelszavával indítani, akkor ahhoz extra jogokat kell biztosítanunk, hogy létre tudják hozni a gépük accountját a tartományban és hogy be is tudják léptetni azt. Amennyiben mi egyáltalán nem készítjük elő az előbbiekben leírt módon a gépeket, akkor ahhoz, hogy ezt megtehessék, a következő jogokra van szükségük az OU-n, amely az újonnan létrehozott gép azonosítóját tárolja:

- Olvasási jog
- Számítógép-objektumok létrehozása

Amennyiben előkészítettük az ügyfélgépek azonosítóit, akkor az egyes gépazonosítókhoz kell a következő jogokat rendelni annak érdekében, hogy a felhasználó azt a saját azonosítójával később telepíteni is tudja:

- Az ügyfélazonosító jelszavának változtatása/beállítása
- A számítógépobjektum minden tulajdonságának olvasása/írása *(Read/write all properties on the computer object)*
- A számítógépobjektum jelszavának változtatása/beállítása.

Ezeket a jogokat nem az objektumot tartalmazó OU-hoz hanem magához a gép azonosítójához kell rendelni!

Ez lehetővé teszi a felhasználó számára, hogy telepítse rendszerét, de csak akkor, ha egy rendszergazda előkészíti a gépazonosítót. Tudom, az is elég ritkán fordul elő, hogy a felhasználóra bízzuk a saját gépének telepítését, de itt lehet bátran, hisz erre találták ki. Ne kelljen előtte ülni, CD-ket adagolni időről-időre, egyszerűen lehet indítani. S ha átszabjuk a varázsló oldalait *(erről lesz szó a következő részben)*, akkor magyarul beszél a felhasználóhoz. Minden adott ahhoz, hogy valóban a legkevesebb beavatkozással telepítsük az ügyfélgépeket. Ám nem szeretünk kiadni felhasználóknak extra jogokat, ezért aztán jó megoldás az is, ha létrehozunk az AD-ban egy olyan felhasználót, akinek semmi más joga nincs, mint a RIS-sel telepített gépeket beléptetni a tartományba, ha azt a rendszergazda előkészítette. Majd amikor telefonon keresztül navigáljuk a felhasználót, hogyan is indítsa el a telepítőt, akkor egy ártatlan felhasználó azonosítóját kell megadnia, nem a sajátját ahhoz hogy elinduljon a telepítő. Így a rendszergazdának is a lehető legkevesebb a dolga, ugyanakkor nem hoztunk létre újabb biztonsági rést a pajzson, amely így is eléggé lyuggatott szegény.

Dorner Csilla
MCSE

A Windows NT és a Windows 2000 memóriakezelése – *más megközelítésben*

A Tech.Net tavaly szeptemberi számában Fóti Marcell ezzel a címmel írt cikket a Windows memóriakezeléséről. Az írás az én ízlésem szerint nagyon Windows-pártira sikeredett, amit nem mulasztottam el a szerző tudomására hozni.

A Windows NT és a Windows 2000 memóriakezelése

Kölcsönös anyázások után győzött a józan ész: kiderült, hogy mindketten irtózunk a szélsőségektől. Egyikünk sem képzeli azt, hogy minden számítástechnikai haladás a Windows-tól ered, de azt is bosszantónak tartjuk, amikor egy Linux-hívő a Windows-t gyalázza, miközben kedvenc operációs rendszerén a Windows-emulátor fut. Végül abban is egyetértettünk, hogy a Windows nem szorul olyan dicséretre, amely nem a valódi érdemeit emelik ki, ezért kaptam lehetőséget kritikai észrevételeim publikálására. Nem szeretném pontról pontra elemezni a szóban forgó cikket, inkább saját logikám szerinti megközelítésben világítanám meg a vitatott részeket.

Az operációs rendszer és a memóriakezelés

Amíg egy gépen csak egyetlen program fut *(ez általában az operációs rendszer)*, övé az összes erőforrás. De ha egy második alkalmazást is elindítunk, akkor osztozniuk kell többek közt a memórián is. Ez az osztozás nem testvériesen történik, hanem szigorú alá-fölérendeltségi alapon. Az operációs rendszer átadja az alkalmazásnak a szükséges erőforrást *(feltéve, hogy van miből)*, a program pedig futni kezd. Az elindított kód azonban nem mindig csak a számára kiutalt területen matat, rossz szándékú, vagy rosszul megírt alkalmazások megpróbálhatnak belepiszkálni a mások által használt memóriatartományokba is. Az operációs rendszer feladata, hogy megvédje magát, ill. biztosítsa, hogy a párhuzamosan futó több száz végrehajtási szál tényleg egymástól elkülönített memóriaterületeket használjon.

Csakhogy az operációs rendszer nem láthatja előre az egyes programok szándékát, így segítség nélkül nem boldogulna. Segítséget pedig csak a processzortól kaphat, attól az eszköztől, amelynek módjában áll megvizsgálni minden utasítást, mielőtt végrehajtaná. A processzorgyártók pedig kifejlesztették a multitaszkos környezetet támogató *(védett módban futó)* processzorokat, amelyek észreveszik, ha egy program tiltott területre téved, és erről értesítik az operációs rendszert. Az operációs rendszer pedig eldönti, hogy mi a teendő. Ebből persze az is nyilvánvaló, hogy védett módban a processzor nemcsak az egyes memóriaterületeket tudja megkülönböztetni, hanem az ott futó kódok jogait is figyelembe tudja venni. A supervisor privilégiumú operációs rendszernek több mindent megenged, mint a felhasználói programoknak.

Virtuális memóriakezelés

Eddig az operációs rendszer abból gazdálkodott, ami fizikailag a gépben volt. A processzor intelligens memóriakezelésének és némi háttértárnak a felhasználásával azonban mód nyílik nem létező memóriaterületek felhasználására is. Ezt az úgynevezett virtuális memóriát – amelyből egy rész a fizikai memóriában tárolódik, a többi meg a merevlemezen létrehozott fájlban –, a programok közt szét lehet osztani, aztán szükség szerint csak cserélgetni kell a tartalmakat. Ha ráadásul a virtuális memória azonos méretű, mondjuk 4k-s lapokra van tagolva, akkor a csereberélés is egyszerűbb: a fizikai memóriában ideiglenesen nélkülözhető lapokat kilapozhatjuk fájlba, a pillanatnyilag szükségeseket meg belapozhatjuk onnan a felszabaduló helyre. *(Gyakran a háttértár gyorsítása érdekében az operációs rendszer a fizikai memória egy részét disk-cache céljára használja, ilyenkor előfordulhat, hogy a kilapozás valójában egy másik memóriaterületre történik. Néha belapozni is lehet onnan, feltéve, hogy a lap még a cache-ben van, általánosságban azonban a lapozás lemezművelettel jár.)*

Nem számít, ha a kód/adat/stack nem összefüggő fizikai memóriaterületre lett betöltve, mert a lapok logikai sorrendjének nyilvántartása alapján laphatár-átlépéskor a processzor az alkalmazást átugratja a fizikailag teljesen máshol elhelyezkedő folytató lapra. Ebből a program semmit nem vesz észre, és úgy fut tovább, akár egy folytonos tartományon. Ha pedig olyan lapra történik hivatkozás, amely nincs a memóriában, a processzor laphibát jelez, az operációs rendszer meg rögtön lapoz.

A flexibilis memóriakezelés ára viszont a lapnyilvántartás nyűgje, és a belőle adódó időveszteség. 4k lapméretnél és 4G virtuális memóriánál egymillió lapot kell nyilvántartani. Ehhez az operációs rendszernek *(Intel processzor esetén)* kétféle táblázatot, egy PageDirectory-t és 1024 Page Table-t kell vezetnie, ami észrevehetően, de azért nem drámaian rontja a rendszer teljesítményét.

Alternatív memóriakezelés

Az Intel processzorai kínálnak egy másik védett módú memóriakezelést is, a szegmentálást. Ez éppen úgy biztosítja, hogy a párhuzamosan futó végrehajtási szálak egymástól elkülönített memóriaterületeket használjanak, mint a VM, csak itt nincs lapozás. Az alkalmazás teljes egészében betöltődik, adatainak memóriaszegmenset allokál az operációs rendszertől, aztán teszi a dolgát. Ez a memóriakezelés nyilvánvalóan gyorsabb programfutást tesz lehetővé, mint a lapozós.

Ám ha a programok cserélődnek a memóriában, akkor már nem is olyan előnyös ez a mód. Az egyes alkalmazások ugyanis különböző nagyságú egybefüggő memóriaterületeket igényelnek. A leálláskor felszabaduló tartományok vagy kicsik, vagy túl nagyok a

következő induló alkalmazásnak, azaz a memória töredezik. Ilyenkor jön a memória töredezettség-mentesítő Garbage Collector.

Melyik a korszerűbb?

Mindkét memóriakezelésnek vannak előnyei és hátrányai, és egy operációs rendszer fejlettségét nem az határozza meg, hogy ezt, vagy amazt a memóriakezelést használja. Lényeg, hogy a számára optimálist részesítse előnyben.

A processzor által támogatott memóriakezelésből a Windows a lapozós technikát választotta *(szeretném hangsúlyozni, hogy a VM nem Microsoft, de még csak nem is Intel találmány, használták már a Windows előtti időkben is)*. A választás nem meglepő, a Windows esetében a virtuális memóriának van előnye.

A Windows szerverekkel ellentétben a Novell szerverei dedikált szerverek, azaz munkaállomásként nem használhatók. Ha tehát a NetWare-t elindítjuk, betöltődik az összes szükséges program *(a memóriaigény bájtra kiszámítható)*, aztán jó ideig nem kell a konzolhoz ülni. Mi haszna lenne itt a lapozásnak? Viszont tapasztalatból tudom, hogy vannak *(nem a Novell által fejlesztett)* alkalmazások, amelyek párszori újraindítása szinte „megöli" a NetWare-t: az annyira lelassul, hogy jobb restartolni. *(Világos, hogy ez nem a Novell hibája, de attól még kellemetlen bír lenni.)*

Az, hogy a Novell is kezd áttérni a lapozásra, számomra nem a VM „magasabbrendűségét" mutatja, hanem azt jelzi, hogy a NetWare szerver egyre kevésbé tekinthető dedikáltnak, ami a „legendás" stabilitást is érintheti. *(Megjegyzem, hogy dedikált szerverként extrák nélkül használva a Windows NT-vel hasonlóan jó tapasztalataim voltak, mint a Novellel. Évekig hiba nélkül futott, persze jóféle vason.)*

Na, akkor most hány mega kell?

Az eredeti cikk arra a kérdésre kereste a választ a rendszereszközök *(Task Manager, Performance Monitor)* segítségével, hogy mekkora egy adott terheléssel futó Windows szerver fizikai memóriaigénye. A Windows-nak van minimális fizikai memóriaigénye, ez szerepel a termék dobozán. Minden kliens csatlakozása igényel további tárat. Bármely alkalmazás hozzávetőleges igénye becsülhető *(biztosan nem nagyobb, mint a kód, valamint az általa használt stack és adatterület együttes mérete)* is. Ezek a plusz igények azonban a virtuális memóriára vonatkoznak, amit az operációs rendszer dinamikusan képez le fizikai memóriára. A Microsoft fejlesztési filozófiája a hardver teljes kihasználását célozza meg, ezért az bizton állítható, ha van bőven memória, akkor bent lesz az egész, ha meg nincs, akkor darálja a merevlemezt. Az viszont, hogy a darálás mekkora fizikai memóriánál kezdődik, legfeljebb kísérletileg határozható meg. Ráadásul, hogy a rendszer mikor fut elfogadható sebességgel, még szubjektív megítélés kérdése is.

Amikor valaki számítógép-konfigurációt specifikál, van preferált alaplap- és processzorgyártója, CD-olvasó- és merevlemezmárkája, egyáltalán minden komponensről meggyőződéssel meg tudja mondani, hogy melyik típus a tuti, csak a memóriamodulokról beszél mindenki kizárólag méretük alapján. Pedig a Windows nagyon intenzíven használja a memóriát, így egyáltalán nem mindegy, hogy mennyire megbízható az, ami a gépünkben van. Tudni kell, hogy a memóriaműveletek, különösen az írás nagyon precíz összhangot igényel az alaplap, és a memóriamodulok között. Erre a pontos együttműködésre az olcsó RAM-ok nem mindig képesek tartósan, ez pedig az egész rendszer stabilitására is kihat, ami különösen egy szervernél kínos. *(A NetWare nem ennyire kényes, hiszen ott nincs intenzív lapozás.)* Tapasztalatból mondom, hogy a Windows lefagyások túlnyomó részét a nem megfelelő memóriaműködés okozza. Ne feledjük tehát, az azért nem az operációs rendszer hibája, hogy valakit a hardverkereskedő „begagyizott". *(Lásd még: Farkasokkal Netmonozó, 8. oldal. A szerk.)*

Tóbiás Ferenc
ftobias@elender.hu

SPAM – jó vagy rossz?

Lehetséges válaszok

A spam mibenléte, az erre vonatkozó eddigi, jelenleg hatályos illetve a mindjárt hatályba lépő szabályozás alapos áttekintése megérdemel annyi energiát az olvasótól, amennyit az elolvasása igénybe vesz, mivel a jogi megközelítés változása igen tanulságos az Internet jogi szabályozásának egészét tekintve is.

A spam kérdése, előnyei vagy hátrányai, megengedhetősége, kezelése az utóbbi időkben egyre nagyobb vihart kavart a Lajtán innen és túl is. Itthon a december utolsó napjaiban az Országgyűlés által - a több bizottsági fordulóban benyújtott módosító indítványok, és a tervezet alapos átgyúrása után - végül a kormánypártok és az ellenzéki pártok együttes támogatásával elfogadott elektronikus kereskedelmi törvénnyel kapcsolatban került előtérbe a kérdés, míg az Európai Unióban a készülő új adatvédelmi irányelv során „dobták fel" a labdát. Úgy tűnik, hogy a kérdés mindkét jogi normában egybecsengő rendezést nyert, így a jelen írás megjelenésekor már jogtörténeti jellegű lesz, hiszen a témát rendező jogi szabályozás 2002. január 23-án hatályba lép.

Mi a spam? Ide sorolunk-e minden olyan e-mailt, amely postaládánkba naponta beesik, és a legkülönbözőbb dolgokra próbálja figyelmünket irányítani? A fogalom a kialakult egységes megközelítés szerint nem minden ilyen e-mailt sorol ide, csak a kéretlen, üzleti célú, többnyire tömegesen, elektronikus úton elküldött üzeneteket. Tehát a pusztán figyelemfelhívó, pl. vallási térítési szándékkal küldött e-mail nem tartozik ebbe a körbe. Így az egyik sokat vitatott egyház által küldött kéretlen tömeges e-mail nem spam, és amint látni fogjuk, hatályos jogi szabályos sincs és a közeljövőben nem is lesz az ilyen jellegű e-mailekre. **A spam tehát csak az, ami üzleti célból érkezik, és anélkül, hogy ezt kérném.** *(Amint látni fogjuk, a jogalkotás ugyancsak megkérdőjelezte a spam kéretlenségét, tehát gyakorlatilag az üzleti célú e-mailre szűkült a fogalom.)*

A spam azért vált meglehetősen gyorsan közkedveltté a reklámozók körében, mert küldője számára olcsó, hatékony, gyors és nagy tömegben juttatható el a célszemélyekhez. Egy kutatás szerint, amíg a hagyományos direkt marketing egy küldeményre lebontott költsége 0,5-1 USD, addig az e-mailé 10 cent, ezzel szemben a direkt marketing hatékonysági mutatója a 0,5-2%, az e-mailé viszont 5-15%. *(Forrás: dr. Jóri András, Unsolicited Commersial Communications and DP. EU Commission, Internal Market DG, 2001.) (Ha ez így volna...! A szerk.)* Kétségtelen tehát, hogy mennyivel előnyösebb az elektronikus levél a reklámozó számára, mint a hagyományos direkt marketing. És akkor még nem is beszéltünk arról, hogy mennyivel könnyebben és gyorsabban juttatható célba az e-mail, mint a hagyományos postai levél. A spam melletti érvek bővülnek még azzal is, hogy itt nem termelődik olyan mértékű papírszemét, mint a hagyományos reklámnál, gondoljunk csak arra a tarka papírhalomra, amely a lépcsőházakban keletkezik hetente a reklámújságokból. A spamnek azonban van olyan hátránya is, amely a klasszikus papírreklámnak nincs: itt a költség jelentős része a címzettnél keletkezik, aki ugye rendszerint nem kéri, sőt rögtön a virtuális szemétládájába továbbítja a reklámokat. Bizony bosszantó, amikor a még igen drága telefondíj, sőt Internethozzáférési díj gyors ketyegése alatt töltődnek le a terjedelmes kéretlen reklámok. Még ennél is komolyabb ellenérv a spammel szemben azonban az, hogy egy bizonyos mennyiségi határ átlépését követően olyan mérvűvé válhat, amely magára az Internetszolgáltatásra, a hálózat biztonságos működtetésére is veszélyes lehet. Egyes beszámolók szerint nem ritka a címzettekhez beérkező napi 30 darab spam sem, amelyek érdekes módon jellemzően távol keleti, különösen koreai eredetűek.

Az érvek és ellenérvek, a különböző lobbiérdekek érvényesülése ezidáig két fő megoldási rendszert alakított ki: az **opt out** és az **opt in** rendszereket. Az opt out megoldás lényege az, hogy a kéretlen üzleti célú levelek küldése mindaddig megengedett, amíg a címzett egy erre biztosított módon nem jelzi, hogy a jövőben nem kíván ilyen jellegű üzenetet kapni. Az opt in rendszer pont fordítva, csak akkor teszi lehetővé üzleti célú levelek küldését, ha a címzett előre jelzi, hogy kíván ilyen jellegű üzeneteket kapni. A kétféle rendszer kialakulása megelőzte az e-mailek tömeges elterjedését: először a hagyományos marketing egyes formáira adott válaszként jelentek meg az európai és a magyar jogalkotásban.

Az Európai Unióban a telekommunikációs szektorban a személyes adatok és a magánélet védelméről kiadott 97/66/EC Irányelv volt az, amely először rendelkezett az automatikus telefonhívás és fax vonatkozásában - természetes személyek esetében - az opt in rendszer alkalmazásáról. Más esetben, tehát a direkt marketing más, nem automatikus formájánál, illetve nem természetes személy címzettnél a tagállami szabályozásra bízta az opt in/opt out közötti választást. Az elektronikus kereskedelemről szóló 2000/31/EC Irányelv az információs szolgáltatások keretében szolgáltatott üzleti célú üzenetekkel szemben minimális követelményként azt írta elő, hogy magából az üzenetből egyértelműen ki kell derülnie az üzleti célnak, és mind az áru eladójának, vagy a szolgáltatás nyújtójának, mind az ajánlat jellegének egyértelműnek és azonosíthatónak kell lennie. Az Irányelv az opt out rendszert preferálta a természetes személyek vonatkozásában, és nem szólt a jogi személyekről.

A két elemzett szabályozás tükrében meglepő módon mégis előretört az opt in rendszer, és a 15 uniós tagállam közül egyharmadában ezt a szabályozást alkalmazták. Így ma opt in rendszer szerint lehet direkt marketinget elektronikus úton folytatni

Ausztriában, Dániában, Finnországban, Németországban és Olaszországban. Az Európai Unió egységes belső piacának elvét szem előtt tartva elgondolkodtató, hogy az a direkt marketinget folytató cég, amely Franciaországban került bejegyzésre az opt out szerint működhet, míg a Németországban működő versenytársnak az opt in rendszer miatt sokkal nehezebb a dolga. A luxemburgi bíróság elé tudomásunk szerint nem került ilyen kereset, de érdekes lenne egy olyan bírói döntés, amely az ilyen megkülönböztető tagállami rendelkezést nem minősítené a belső piac egységességét, az áruk és szolgáltatások szabad áramlását akadályozó tényezőnek. Ám úgy tűnik, hogy az Európai Unió meghaladva az eddig szabályozást, fel kívánja számolni a tagállami „kétféleséget" a direkt marketing és így a spam szabályozása területén. Az Európai Parlament és a Tanács közös irányelve az adatvédelemről a sok vita eredményeként az opt in rendszert preferálja szemben az opt outtal. Érdekes módon a kontinentális országok forszírozták az opt in rendszer egységessé tételét a konzervatívnak tekintett Angliával szemben, amely az opt out mellett kardoskodott. Anglia azonban fontosnak tartotta, hogy az adatvédelmi szabályozás keretében váljék lehetővé az Internetezők adatainak biztonsági okokból átmeneti időre történő tárolása. Ez ellen azonban éppen a kontinentális országok jogvédő szervezetei léptek fel erőteljesen. A kompromisszum azonban megszületett: Európában általánossá válik az opt in, míg megengedett az Internetezők adatainak átmeneti időre történő tárolása. *(A kérdés csak az, hogy mennyi az az átmeneti idő? De az ezen való töprengés egy másik cikk témája lehet.)*

Az Európai Unióba még nem tartozó, de annak szabályozásához szükségszerűen alkalmazkodni kénytelen Magyarországon jelenleg négy hatályos jogszabály rendelkezik közvetlenül vagy közvetve a spam-ről, és egy, 2001. december 24.-én kihirdetett, de hatályba csak **2002. január 23-án** lépő törvény tűnik rendezni egyértelműen a kérdést.

A személyes adatok védelméről és a közérdekű adatok nyilvánosságáról szóló 1992. évi LXIII. törvény értelmében személyes adat a meghatározott természetes személlyel kapcsolatba hozható adat, az adatból levonható, az érintettre vonatkozó következtetés. A személyes adat az adatkezelés során mindaddig megőrzi e minőségét, amíg kapcsolata az érintett természetes személlyel helyreállítható. Az adatvédelmi biztosnak az Internettel kapcsolatban kiadott állásfoglalása értelmében a törvény logikus értelmezés szerint az e-mail cím és az IP cím *(sic!)* is személyes adatnak minősül. Ez azt jelenti, hogy ezen adatok kezelése során be kell tartani mindazokat a szabályokat, amelyeket a törvény előír. Így az e-mail címet csak rendeltetésének megfelelően lehet felhasználni - és a spam nem tartozik ebbe a körbe, tehát a spam küldése céltól eltérő adatkezelést valósít meg. Ne felejtsük el, itt ismét csak természetes személyekről beszélhetünk, vagyis jogi személyek, cégek, szervezetek vonatkozásában a védelem nem érvényesül. Érdekes kérdés persze, hogy a személyes, de egy cégnek delegált domain név alatt működtetett e-mail vonatkozásában mi állapítható meg? Ugyancsak nem hagyható figyelmen kívül, hogy ha vizsgálódásunkat kizárólag adatvédelmi szempontból folytatjuk, akkor céltól eltérő adatkezelésnek minősül az a kéretlen üzenetküldés is, amely nem üzleti célú, hanem más okból került elküldésre. Emlékezzünk a cikk elején említett vallási felekezet térítő jellegű küldeményeire, amelyek e törvény alapján tiltottak! Mindezek persze csak abban az esetben igazak, ha az e-mail cím megszerzése során sértik meg az adatvédelmi törvény előírásait. Abban a - kevéssé valószínű - esetben, ha az e-mail címre véletlen generáció során tesz szert a felhasználója, a céltól eltérő adatkezelés nem állapítható meg. Az adatvédelmi szabályok megsértése egyébként igen súlyos, adott esetben büntetőjogi szankcióval is járhat, így az ilyen lépések bizony megfontolandók! A kutatás és a közvetlen üzletszerzés célját szolgáló név- és lakcímadatok kezeléséről szóló 1995. évi CXIX. törvény lehetővé teszi, hogy direkt marketing céljára a felhasználó a nyilvántartásból üzleti célra adatot igényeljen, és egyidejűleg előírja egy tilalmi lista felállítását, amelyen azon érintettek név- és lakcímadatai szerepelnek, akik nem járultak hozzá, hogy személyes adataikat e törvényben meghatározott közvetlen üzletszerzési célok valamelyikére felhasználják, vagy megtiltották azok e célból történő további kezelését. Igen, kedves olvasó, jól gondolja: ez az opt out rendszer! Viszont a törvény csak a név- és lakcímnyilvántartásra vonatkozik, így sajnos itt csak az analógia miatt került sor a megemlítésére, e-mail címekre nem alkalmazható.

A gazdasági reklámtevékenységről szóló 1997. évi LVIII. törvény - az Európai Unió elektronikus kereskedelemről szóló irányelvének ismertetett rendelkezéseihez hasonlóan - a reklám ilyen mivoltának azonosíthatóságát, a reklámozó egyértelmű *(megnevezése, székhelye megjelölése, adószáma feltüntetése)* meghatározását írja elő, tovább azt, hogy a **reklámot csak a környezetétől elkülönítve szabad közzétenni.** A jogszabály nem tesz különbséget a reklámhordozó vagy a reklámot továbbító közeg között, így az off line vagy on line médiára, az elektronikus közegre vagy a hagyományos hirdetőoszlopon közzétett reklámra, tehát a spamre is vonatkozik. Kicsit előreszaladva a mondanivalóban: itt szükséges jeleznem, hogy az elektronikus kereskedelemről elfogadott magyar törvény nem érinti ezeket az előírásokat, tehát ezek továbbra is hatályban maradnak és vonatkoznak valamennyi elektronikus úton küldött reklámra, azaz spam-re.

A távollévők között kötött szerződésekről szóló 17/1999. *(II. 5.)* Korm. rendelet, amely egyaránt rendezni kívánta a katalógusáruházakból, a direkt marketing útján, az off line és az on line médián keresztül történő értékesítést is, a fogyasztó kifejezett hozzájárulását írja elő ahhoz, hogy a gazdálkodó szervezet a szerződéskötés céljából automata hívókészüléket, illetve távmásolót *(telefaxot)* használjon. A kissé nehezen érthető megfogalmazás mögött az Európai Unió ismertetett 97-es irányelvével összhangban álló opt in rendszer fedezhető fel az automatikusan *(emberi közreműködés nélkül)* tárcsázott telefonhívások vagy küldött faxok esetében. A rendelet egy másik fordulata azonban minden más esetre, vagyis olyan közvetlen kapcsolatot lehetővé tevő távközlő eszközre, amely nem automata telefonhívás vagy fax, lehetővé teszi, hogy a fogyasztó kifejezett tiltakozása hiányában *(vagyis az opt out rendszerre alapítva)* a gazdálkodó szervezet ilyet használjon. Ez az a rendelkezés, amely ténylegesen jogszerűvé teszi ma a spam küldését, természetesen figyelemmel az e-maileket, mint személyes adatokat értelmező adatvédelmi biztosi állásfoglalásra, és az ebből következő törvényi korlátokra.

Az év végi nagy hajrában, a jogalkotási dömping keretében fogadta el az Országgyűlés, és a Magyar Közlöny 2001. december 24-i számában karácsonyi ajándékként a fa alá is került **az elektronikus kereskedelemi szolgáltatások, valamint az információs társadalommal összefüggő szolgáltatások egyes kérdéseiről** hangzatos címet viselő 2001. évi CVIII. törvényt. A törvény részletes elemzésére egy másik cikk keretében kerítünk

sort, ezúttal csak a spam kérdésének vizsgálata szempontjából ismertetjük. A 14. § az információs társadalommal összefüggő szolgáltatás felhasználásával küldött reklámokra vonatkozó különös szabályok cím alatt megismétli a reklám azonosíthatóságára vonatkozó kívánalmat, és határozottan úgy rendelkezik, hogy kizárólag az igénybevevő *(értsd: címzett)* **egyértelmű, előzetes hozzájárulásával** küldhető elektronikus úton, levelezés során reklám. Tehát a jogalkotási folyamat során a kezdetben a törvény tervezetében szerepelt, és az előterjesztőt képviselő, a törvény szövegét megfogalmazók által oly harciasan védett opt out rendszerrel szemben **győzedelmeskedett az opt in.** A továbbiakban előírja a törvény, hogy nyilvántartást kell vezetni azokról, akik kívánnak ilyen jellegű küldeményeket kapni, és nem szabad olyanok számára reklámot küldeni, akik a meghatározott nyilvántartásban nem szerepelnek. Azok számára is lehetővé kell tenni, hogy a továbbiakban megtiltsák, hogy részükre reklámot küldjenek, akik ezt korábban igényelték, így tájékoztatni kell a címzettet arról az elérhetőségről, ahol a reklámok küldésének megtiltása iránti igényét bejelentheti. A tilalom a reklámozó, a reklámszolgáltató, illetve a reklám közzétevője által küldendő összes reklámra vonatkozik. Az opt in rendszer tiszta és körültekintő szabályozásával állunk tehát szemben. Az érdekes csupán annyi, hogy amíg a 14. § az elektronikus levelezés útján történő reklámokról rendelkezik, addig a törvény hatályát meghatározó 1. § kifejezetten kiveszi a törvény hatálya alól az elektronikus magánlevelezést. E logikai rend szerint tehát **a spam üzleti célú levelezés.** Vagyis az opt in rendszer csak az üzleti célú reklámra vonatkozik, és amint említettük, a nem üzleti célú, de kéretlen és tömeges e-mail küldésre nem. Az pedig majd a jogalkalmazás dolga lesz, hogy meghúzza a vonalat: meddig üzleti a cél és mikortól nem az.

További érdekes hézagnak látszik, hogy amíg a frissen elfogadott törvény csak az elektronikus levelezésre vonatkozik, addig a kéretlen üzleti célú üzenetek már újabb eszközt: a mobiltelefont kezdik felfedezni maguknak. Erre, vagyis az sms-re pedig nem vonatkozik a XXI. század első évének végén elfogadott törvény. Pedig nincs messze az a jövő, amikor majd kéretlen sms-ek lepik el mobilunkat és zavarnak meg napi tevékenységünk közben sokkal nagyobb akadályt okozva, mint a kéretlen e-mailek. A Mobilpont már be is vezette a multilevel-marketing rendszerben felépített hálózatban, hogy az oda beszervezett vállalkozók, akik készek reklámtartalmú sms-eket fogadni, minden egyes ilyen üzenet után 5 forintot kapjanak. A horog tehát a vonzó csalival már ott lóg a piac tavában, és már jócskán akadt arra ráharapó hal is, és még nagyobb a még csak pedzegetők száma. Nincs azonban már messze az a jövő, amikor a horgon már nem lesz csali sem, viszont az egész mobilozó társadalom hízott halként lóg majd rajta. De hát a konzervatív jogalkotás mindig az események után kullog, így a kérdés rendezése majd pár év múlva egy másik jogszabályban fog megtörténni.

Dr. Mayer Erika
IKRP Nádas & Mayer Ügyvédi Iroda

Portszűrés IPSec-kel

Munkánk során sokszor kerülünk olyan helyzetbe, hogy egy kiszolgálót *(legyen a feladat web, levelezés vagy bármi más)*, meztelenül, pőrén kell az Internethez csatlakoztatni. Sehol egy jó kis tűzfal, marad az önvédelem. Az már alapszabály, hogy minden szolgáltatást, portot, amire nincs szükség, be kell zárnunk, de mégis, hogyan?

TCP/IP Filtering vs. IPSec Filtering

A szolgáltatások leállítása nem 100%-osan biztonságos, valahogy a hálózati forgalmat kellene szűrni. A Windows NT már régóta lehetővé teszi az IP forgalom szűrését, bár elég szigorú és kényelmetlen korlátozásokkal. A Windows 2000 beépített IPSec támogatása – bár nem erre tervezték – ugyancsak képes a hálózati forgalom szétválogatására, mégpedig jóval rugalmasabban, mint az egyszerű TCP/IP szűrés. Íme egy összehasonlító táblázat:

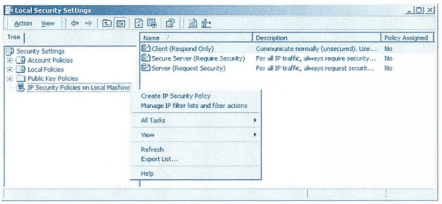
Szolgáltatás	IPSec szűrés	TCP/IP szűrés
Hatókör	Megadott IP címek közötti forgalom	Minden kártya, minden IP cím
Forrás IP cím	A szabályokban beállítható	Szűrés csak portcím alapján
Újraindítás szükséges?	Nem, a hatás azonnali *(!)*	Igen
Szűrhető protokoll	Bármi, ami IP	Csak TCP, UDP, ICMP
Válasz a blokkolt csomagokra	Nincs válasz	Reset csomag

Főbb különbségek az IPSec házirendekkel és a TCP/IP beállításokkal történő csomagszűrés között

Emellett az IPSec házirend – szükség esetén – Windows 2000 tartományban Group Policy-vel is beállítható, illetve az IPSec házirendek parancssori eszközzel is konfigurálhatók *(ehhez a Resource Kitben található ipsecpol.exe-re van szükség)*. A cikk keretében a grafikus felhasználói felület használatát mutatjuk be, ráadásul nem a Group Policy-n, hanem a gépenként külön-külön használható Local Security Policy-n, a helyi biztonsági beállításokon keresztül.

A Local Security Policy

A Local Security Policy a Windows 2000 *(Professional is)* helyi biztonsági beállításait tartalmazza. Az eszközt az Administrative Tools menüben találjuk meg *(ez a menü Professional-on csak akkor látszik, ha a TaskBar tulajdonságai között külön engedélyezzük)*.



A Local Security Policy ablaka, benne az IPSec beállítások

Ami nekünk érdekes, az az „IP Security Policies on Local Machine" csomópont, illetve az alatta található elemek.

Néhány szó az IPSec házirend működéséről

Mint azt már említettük, az IPSec eredeti célja nem a csomagszűrés, ezért a cikk során nagyon sok mindentől eltekintünk majd, ami esetleg a varázslókban szembe jöhet. A lényeg: az IPSec Policy Agent nevű rendszerszolgáltatás minden érkező IP csomagnál ellenőrzi az aktuálisan érvényes IPSec házirendet *(az előző ábra jobb oldali részén a három alapértelmezett házirend látható, de egyik sincs érvényesítve)*. Az IPSec házirend *(IPSec Policy)* IPSec szűrőlisták *(IPSec Filter Lists)* és szűrőműveletek *(Filter Actions)* összerendeléséből áll. Egy házirend később több szűrőlistát és műveletet is összerendelhet.

Az IPSec Filter List egy vagy több kommunikációs csatorna *(feladó, címzett IP címe, protokoll, feladó, címzett portcíme)* listája. A Filter Action pedig vagy Engedélyez *(Permit)* vagy Tilt *(Block)* vagy pedig bizonyos titkosított kapcsolatot épít fel *(Negotiate Security)*. Ezek közül mi az utóbbit nem használjuk, a szűrőakciókat tisztán a csomagok tiltására és engedélyezésére használjuk majd fel. Világos tehát, hogy a csomagszűrés beállítása sorrendben a következő műveletekből áll:

- IPSec szűrőlisták definiálása
- Szűrőakciók definiálása
- Szűrők és akciók összerendelése: az IPSec házirend létrehozása.

Az első két műveletet közös dialógusablakban végezhetjük el, amit *(az előző ábrán is látható menüben)* a „Manage IP Filter Lists and Filter Actions" parancsra kattintva nyithatunk meg.

IPSec szűrőlista létrehozása

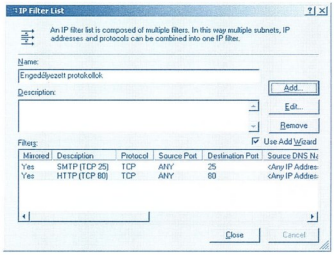
Összesen két szűrőlistát kell létrehoznunk:

- Egyet, ami az összes protokollt tartalmazza, amit majd engedélyezni akarunk
- Egy másikat, ami minden bejövő protokollt tartalmaz, ez fogjuk majd tiltani.

Nyissuk meg a szűrőlista létrehozására szolgáló dialógust. A listában már két előre elkészített szűrőlistát látunk, az All ICMP Traffic az ICMP forgalom *(pingelés és társai tiltására ezt használhatjuk majd)*, az All IP Traffic pedig a kimenő forgalom definiálására való *(ez utóbbit itt nem fogjuk használni)*. Kattintsunk inkább az Add... gombra, és máris megjelenik az új IPSec szűrőlista definiálására való újabb dialógus! *(A példában olyan szűrőlistát fogunk létrehozni, ami a 25-ös (SMTP) és a 80-as (HTTP) portot tartalmazza)*. Adjunk nevet az új szűrőlistának *(pl. „Engedélyezett protokollok")*, majd kezdjük el felvenni a szűrődefiníciókat *(megintcsak Add... gomb)*. Erre elindul a New IP Filter varázsló.

- Forráscímnek *(Source Address)* válasszuk az „Any IP Address opciót.
- Célcímnek *(Destination Address)* az „A specific IP Address"-t, majd adjuk meg a saját IP címünket. Ha a gépben csak egy hálózati kártya van, választhatjuk a „My IP Address" lehetőséget is.
- Protokolltípusnak válasszuk a TCP-t
- Az IP protokoll portnál a „To this port" mezőbe vegyük fel a kívánt portcímet *(25, illetve 80)*.
- A Finish gomb megnyomása előtt jelöljük be az „Edit Properties" négyzetet.
- A Finish után megjelenő dialógusablak Description oldalán adjunk egy jól értelmezhető nevet a protokollszűrőnek, hibakeresésnél még jól jöhet.
- Végül kattintsunk az OK gombra.

A műveletet ismételjük meg minden egyes protokoll esetén, amit engedélyezni szeretnénk. A végén valami hasonló listát látunk majd:



Az IPSec szűrőlista ablaka, benne a két frissen definiált szűrővel

Ha készen vagyunk, zárjuk be az ablakot, és hozzunk létre még egy szűrőlistát, mondjuk „Minden bejövő forgalom" néven. Ebbe egyetlen egy filtert vegyünk fel, az alábbi beállításokkal:

- Forráscím: Any IP Address
- Célcím: A saját IP címünk
- Protokolltípus: Any

Ha ez is megvan, zárjuk be a dialógusokat, következhet a szűrőakció létrehozása.

Szűrőakció létrehozása

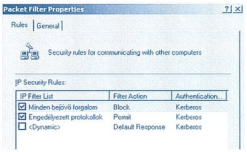
A „Manage IP Filter Lists and Filter Actions" parancsra megjelenő dialógusablak Manage Filter Actions oldalán három előredefiniált akciót találunk. Ezek közül számunkra a Permit érdekes, ami a titkosítatlan forgalmat engedélyezi – ez egyrészt nekünk nagyon megfelel majd. Viszont hiányzik a blokkoláshoz szükséges akció: hozzuk hát létre! Kattintsunk az Add... gombra, erre elindul a Filter Action varázsló. Nevezzük el az új akciót, mondjuk „Block"-nak, majd a „Filter Action General Options" oldalon válasszuk a Block lehetőséget. Ezzel készen is vagyunk, létrehoztuk a blokkoló akciót.

Az IPSec házirend létrehozása

Ezek után már nem is maradt más hátra, mint a konkrét házirend létrehozása. Most válasszuk a már sokat emlegetett menüből a Create IP Security Policy parancsra, erre rögtön megjelenik a külön e célra készített varázsló. Nevezzük el az új házirendet, mondjuk „Packet Filter"-nek, a következő oldalon kapcsoljuk ki az „Activate the default response rule" opciót. Végül hagyjuk bekapcsolva az „Edit Properties" mezőt és kattintsunk a Finish gombra. Megjelenik a már kész házirend egyelőre üres ablaka. Kattintsunk az Add... gombra, erre elindul a „Security Rule" varázsló *(emlékezzünk, most fogjuk az egyes IPSec szűrőlistákat az akcióval összerendelni)*. A varázslóban a következő beállításokat végezzük el:

- A Tunnel Endpoint oldalon hagyjuk az alapértelmezést: „This rule does not specify a tunnel".
- A Network Type oldalon válasszuk az „All network type" lehetőséget.
- Az Authentication Method oldalon hagyjuk az alapértelmezést *(esetünkben nincs jelentősége)*, majd nyugtázzuk az esetleg megjelenő figyelmeztetést.
- Az IP Filter List oldalon válasszuk ki az „Engedélyezett Protokollok" szűrőlistát
- A Filter Action oldalon pedig a „Permit" akciót.

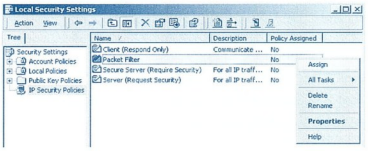
A műveletet ismételjük meg a „Minden bejövő forgalom" szűrőlista és a „Block" akció összerendelésével, és máris készen van a házirendünk:



A frissen létrehozott IPSec házirend

A házirend érvényesítése

Végül nincs más dolgunk, mint nagy levegőt venni, és érvényesíteni a házirendet. Ehhez kattintsunk az MMC konzolban a házirend nevére jobb gombbal és válasszuk az „Assign" parancsot. Vigyázzunk, a korlátozások azonnal érvényre jutnak!



A házirend érvényesítése

Egy jó tanács

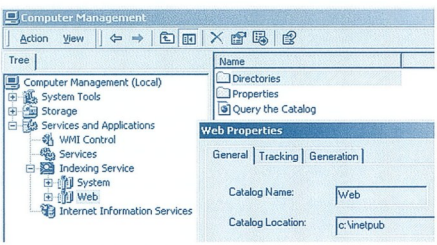
Egy tipikus internetes masina viszonylag ritkán kommunikál kifelé, de előfordulhat *(ha például levelet szeretne küldeni)*. Ha az összes bejövő csomagot blokkoljuk, a gépről kifelé is csak azok a protokollok mehetnek, amelyeket befelé engedélyeztünk *(azaz szűrés nem csatornára, hanem ténylegesen a csomagokra érvényes)*. Ha ezt szeretnénk elkerülni, ne az összes bejövő csomagot blokkoljuk, hanem állítsunk össze egy „feketelistát" a szűrni kívánt protokollokból, és azt a szűrőlistát tiltsuk le. Ez persze kényelmetlenebb megoldás, de kompromisszumokat kötni tudni kell.

Fülöp Miklós
mick@netacademia.net

ASP Suli:
Az Indexing Service

Az Indexing Service szolgáltatás - mely a háttérben csendben indexelgeti a dokumentumok tartalmát *(hogy később azután segítségével kereshessünk közöttük)* – megtalálható a Windows 2000 minden változatában. Ha a szolgáltatás fut, még az egyszerű keresés is felhasználja az Indexing Service-től kapott adatokat, de programozói felületének köszönhetően nagyszerűen használható scriptekből is.

A Windows NT korábbi változataiban külön termékként, az Option Pack-ből telepíthető szolgáltatás az Index Server, mely a Windows 2000-ben az operációs rendszer beépített része lett. Azóta a neve is megváltozott kicsit, így lett Indexing Service. A webes dokumentumok indexelése egyáltalán nem áll távol a szolgáltatás alapcéljaitól, mi sem mutatja ezt jobban annál, hogy már az „üresen" feltelepített Indexing Service is tartalmazza a „Web" nevű alapértelmezett katalógust, valamint jónéhány, webre optimalizált szolgáltatást. Cikkünkben áttekintjük az Indexing Service működését és azt, hogy hogyan használhatjuk ki a szolgáltatás előnyeit webes alkalmazásaink fejlesztésekor.



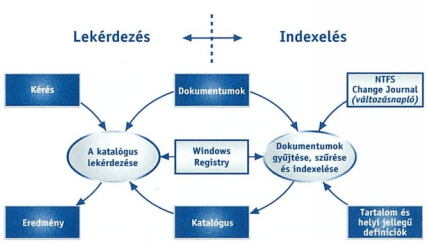
A még „szűz" Indexing Service, és benne az alapértelmezett katalógusok: System és Web

Mi is pontosan az Indexing Service?

Az Indexing Service **[1]** rendszerszolgáltatás, mely helyi és hálózaton keresztül elérhető dokumentumok tartalmát és tulajdonságait indexeli, tárolja. A Windows 2000 szolgáltatása alapértelmezésben az Office dokumentumok *(Word, Excel, PowerPoint, stb.)*, HTML fájlok, MIME üzenetek *(.eml)*, valamint egyszerű szövegfájlok indexelésére képes. Az indexmotor azonban egy jól definiált interfészen *(IFilter,* **[2]***)* keresztül képes gyakorlatilag bármilyen dokumentum indexelésére, ha valaki megírja a hozzá való szűrőt. Ha például a **[3]** címről letöltjük és telepítjük az Adobe PDF IFilter implementációját, az Indexing Service egy csapásra képes lesz .pdf fájlok olvasására és katalogizálására is.

A kigyűjtött információkat azután az Indexing Service katalógusokba szervezi, és mi később ezen katalógusok segítségével kereshetünk az indexelt dokumentumok között. Az Index Server régebbi verzióit COM kompatibilis programozói felület hiányában a webkiszolgálón csak egy speciális ISAPI filter segítségével használhattuk *(ez az .idq, .ida, .htx fájlokat kezelő idq.dll)*. Ez a komponens ma is megtalálható a rendszerben, de jó pár biztonsági rés melegágya, és ma már a legtöbb helyen a használata tiltva van *(vagy kellene, hogy legyen)*. A Windows 2000 Indexing Service szolgáltatása szerencsére már tartalmaz rendes programozói felületet mind a szolgáltatás konfigurálására, mind pedig a katalógusok lekérdezésére – a veszélyes ISAPI filter helyett érdemesebb ezeket használni. Emellett, az Indexing Service tartalmaz egy OLE DB Provider-t is, ami azt jelenti, hogy egy-egy ADO kapcsolat adatforrásaként meghatározhatjuk az Indexing Service katalógusait, és azután ADO műveletekkel is lekérdezhetjük azt. Mi itt a scriptprogramozói felületet használjuk majd, de látni fogjuk, hogy a munkánk során elkerülhetetlen lesz, hogy egy kicsit az ADO objektumait is segítségül hívjuk.

Az Indexing Service működése



Az Indexing Service működésének vázlata

Amint az a fenti ábrából is látható, az Indexing Service feladata kettős: az első feladat a rendelkezésre álló dokumentumok indexelése, a katalógusok előállítása, tekintettel arra, hogy a katalogizálás után a dokumentumokban esetleg bekövetkező változások megjelenjenek a katalógusban is. Ugyanígy reagálni kell az indexelt könyvtárakban később létrehozott dokumentumokra is. Ehhez, amikor csak teheti, az Indexing Service még az NTFS fájlrendszer változásnaplóját is igénybe veszi. Ha pedig ez nem áll rendelkezésre, marad a dokumentumok időnkénti újraindexelése. Amikor az Indexing Service egy dokumentumot indexel, a dokumentumhoz tartozó IFilter szűrő segítségével összegyűjti a fontosabb jellemzőket *(„property"-k: név, cím, méret, stb. – az indexelt jellemzők teljes listáját az Indexing Service MMC konzolban, a kívánt katalógus neve alatti „Properties" pontban találjuk)*. Miután ezzel elkészült, szintén a szűrőtől „elkéri" a dokumentum szöveges tartalmát. A szöveget azután szavakra tördeli (elvileg figyelembe veszi a dokumentum nyelvét, és annak megfelelően át is alakítja a szavakat: megkeresi a szótöveket, általános formákat, stb. *(Például: az „egyelek" szót tartal-*

mazó dokumentum megtalálható lesz, ha az „eszik" szóra keresünk). A magyar nyelvű modul sajnos nem része a Windowsnak, de például a MorphoLogic **[4]** MorphoStem programcsomagja tartalmaz ilyen komponenseket). Miután az optimalizált szólistа megvan, közülük még el kell távolítani azokat a szavakat, amelyek a keresés szempontjából érdektelenek *(például névelők, számjegyek, stb.).* Ezek a szavak *(„zajszavak", „noise words")* nyelvenként ugyancsak változók; az egyes nyelvek zajszavait a System32 könyvtárban, noise.* nevű fájlokban találjuk meg, ahol a * helyén a nyelv azonosítója van *(pl. noise.fra)*. A magyar persze ismét nincs közöttük *(de a MorphoStem azt is feltelepíti)*. A szavakból ezután végre szólisták készülnek, amelyek segítségével végül kereshetünk a katalógusban.

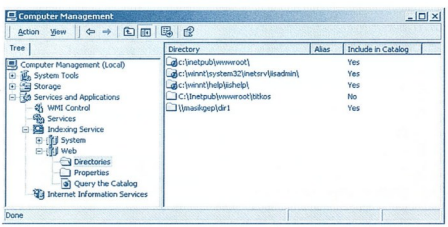
A keresés során a szolgáltatásban található valamelyik katalógusban keresünk, a kérdésünket pedig az Indexing Service által ismert három lekérdezőnyelv valamelyikén határozhatjuk meg. A három lekérdezőnyelv közül az egyik egy SQL alapú, a másik kettő pedig az Indexing Service saját lekérdezőnyelvének két változata *(dialektusa)*. Itt az Indexing Service Query Language *(Dialect 2)* nyelvet fogjuk használni, de akit érdekel a többi, a dokumentációban **[5]** megtalálja.

Kifejezés	Jelentés
alma fa	Az "alma" és a "fa" szavak egyikét vagy mindegyikét tartalmazó dokumentumok
alma AND fa	Az "alma" és a "fa" szavak mindegyikét tartalmazó dokumentumok
"alma fa"	Az "alma fa" szőösszetételt tartalmazó dokumentumok
alma AND NOT fa	Az "alma" szót igen, a "fa" szót viszont nem tartalmazó dokumentumok
alma NEAR fa	Az "alma" és a "fa" szót egymáshoz közel tartalmazó dokumentumok
@size > 20000 & < 100000	Azok a dokumentumok, amelyek mérete 20000 és 100000 bájt között van
@Write > 2001-10-01	A 2001. október 1. óta megváltozott dokumentumok
alma & (@filename = *.pdf)	A .pdf kiterjesztésű dokumentumok, amelyek tartalmazzák az "alma" szót

Néhány példa keresésre

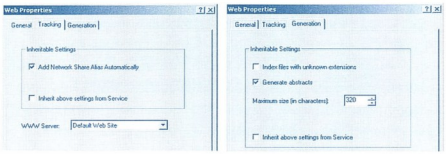
Az Indexing Service használata webes környezetben

Aki hiányolja az ASP példákat, annak biztató hírem van: még egy rövid ideig tartózkodunk a kódolástól, de már közel járunk hozzá. Az Indexing Service-t ugyanis mindenekelőtt fel kell készíteni a webes tartalom indexelésére.



A „Web" katalógusban definiálhatjuk az indexelni kívánt, illetve a kihagyandó könyvtárakat

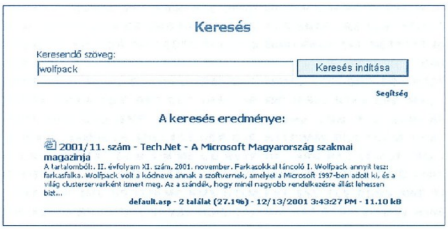
Ne felejtsük el, hogy a keresést mindig egy-egy katalógusban végezzük, ezért érdemes az összetartozó könyvtárakat közös katalógusba gyűjteni. A „Web" nevű katalógus használata nem kötelező, de ha a kódban külön nem állítunk be valami mást, a keresés ebben a katalógusban fog történni. Az ábrán látható, hogy a katalógusokba „negatív" könyvtár is felvehető *(azaz, a könyvtár tartalma ne szerepeljen az indexben)*. Végül vessünk még egy pillantást a katalógus tulajdonságlapjára is:



A katalógusok tulajdonságlapja

Figyeljük meg az első oldal alján található „WWW Server" mezőt! Itt kiválaszthatjuk, hogy az adott katalógus melyik virtuális webkiszolgáló dokumentumait tartalmazza. Erre azért van szükség, mert bár az Indexing Service a fájlokat a helyi *(valós)* elérési úttal katalogizálja *(pl. „C:\InetPub\wwwroot\dir1\default.htm")*, de – ha tudja, melyik webkiszolgálón található a dokumentum – automatikusan tárolja mellé a virtuális elérési utat is *(pl. „/dir1/default.htm")*. Így a találatok megjelenítésekor nem kell külön foglalkozni a korrekt elérési utak kiszámításával.

A másik fontos beállítás a jobb oldali ábra közepén látható „Generate abstracts" opció. Ha ezt engedélyezzük, az Indexing Service minden indexelt dokumentumból *(legfeljebb az opció alatt meghatározott hosszúságú)* szövegkivonatot gyárt és tárol a katalógusban, amit a találatok megjelenítésekor ugyancsak felhasználhatunk.



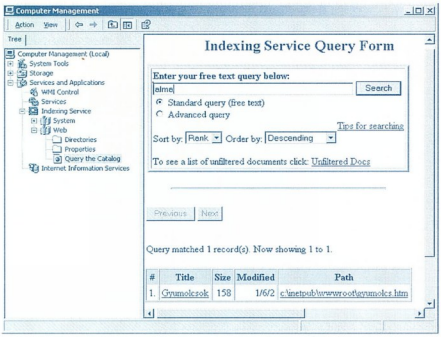
A találatok megjelenítésekor közölhetjük a felhasználóval a találati arányt, a dokumentum lényeges adatait, vagy akár a szövegkivonatot is

Számos lényeges jellemző van még, amit az Indexing Service a katalógusban tárol, ezekre később még ki fogunk térni.

Az Indexing Service COM programozói felülete

Elérkeztünk végre a katalógusokban való kereséshez. Ehhez az Indexing Service COM-kompatibilis programozói felületét, annak is két objektumát használjuk fel majd. Akit érdekel, az Indexing Service teljes referenciája a **[6]** címen érhető el. A két, általunk használt objektum pedig a Query és a Utility lesznek. Az objektumcsalád másik három objektuma *(AdminIndexServer, CatAdm, ScopeAdm)* az Indexing Service adminisztrációjához szükségesek. A példaprogramok a **[7]** címről tölthetők le. Az IndexDemo könyvtár tartalmát másoljuk valahova a helyi lemezre, majd vegyük fel valahova az alapértelmezett webhely alá virtuális map-

paként. Ezután a Indexing Service katalógusában ellenőrizzük, hogy a könyvtárbejegyzés létrejött-e automatikusan. Kis idő múlva már az MMC konzolba beépített kis eszköz segítségével keresni is tudunk majd az indexelt dokumentumok között:



A katalógusok előzetes teszteléséhez elég az MMC konzol

Első lépésként hozzunk létre egy oldalt *(default.asp)*, ami bekér egyetlen keresőkifejezést, majd az elküldi a kereső oldalunknak *(query.asp)*. A lényeges dolgok persze csak ez utóbbi oldalon történnek... lássuk tehát a kódot.

```
sSearch = Request("s") ' Ezt keressük...

Set oQuery = Server.CreateObject("IXSSO.Query")
oQuery.Catalog = "Web"
oQuery.Dialect = "2"
oQuery.Query = sSearch
oQuery.SortBy = "Rank[d], Write[d]"
oQuery.Columns = "DocTitle, Vpath, Path, FileName,
↳ Size, Write, Characterization, Rank, HitCount"

Set oRS = oQuery.CreateRecordset("sequential")

If oRS.EOF Then
  Response.Write "Nincs találat."
Else
  Response.Write "Találatok:<br>"
  While Not oRS.EOF
    For Each oField In oRS.Fields
      Response.Write oField.Name & ": "
      Response.Write oField.Value & "<br>"
    Next
    Response.Write "<br>"
    oRS.MoveNext
  WEnd
End If
```

A kód nagyon egyszerű. Létrehozzuk a Query objektumot *(„IXSSO.Query")*, és kitöltjük néhány jellemzőjét. Ezek:

- Catalog – a keresni kívánt katalógus neve. Ha nem adjuk meg, és IIS van a kiszolgálón, akkor az alapértelmezés is „Web"
- Dialect – a keresőkérdésben használt dialektus azonosítója *(az alapértelmezés is a „2")*
- Query – a keresőkérdés maga
- SortBy – a válaszok rendezési sorrendje. A mezőnevek után írt [d] csökkenő, az [a] pedig növekvő rendezést jelent. A „rank" mező a találati arány mutatója, a többi egyszerű dokumentumjellemző
- Columns – a válaszban visszaadandó mezők listája, vesszővel elválasztva. Itt tulajdonságneveket adhatunk meg. Azt, hogy milyen tulajdonságok léteznek, az Indexing Service MMC modulban, a kívánt katalógus alatti Properties pontban látjuk. Néhány tulajdonságot az alábbi táblázatban is bemutatunk, de akit érdekel, a webkatalógusban használható tulajdonságnevek teljes listájának utánanézhet a **[8]** címen.

Jellemző	Leírás
Characterization	A dokumentum szövegének néhány száz karakteres kivonata
DocTitle	A dokumentum címe *(ha létezik)*
FileName	A fájl neve
HitCount	A találatok száma
Path	A dokumentum fizikai elérési útja *(a fájlrendszerben)*
Rank	Találati arány *(1..1000, a magasabb érték pontosabb találatot jelent)*
Size	A fájl mérete
Vpath	A dokumentum virtuális elérési útja *(a webkiszolgálón keresztül)*
Write	A dokumentum utolsó módosításának dátuma

Néhány dokumentumjellemző és leírása

Miután az alapvető adatokat beállítottuk, meghívjuk az objektum .CreateRecordset() metódusát. Ez a metódus egy paramétert vár, ami „sequential" vagy „nonsequential" lehet, attól függően, hogy egy körben feldolgozandó vagy lapokra bontva, könyvjelzőkkel, több nekifutásban megjelenítendő Recordsetet szeretnénk. A különböző paraméterek hatására létrehozott Recordsetek jellemzői:

	„sequential"	„nonsequential"
Kurzortípus *(.CursorType)*	Forward Only *(0)*	Static *(3)*
Kurzor helye *(.CursorLocation)*	Server *(2)*	Server *(2)*
Zárolás típusa *(.LockType)*	Read Only *(1)*	Read Only *(1)*
Lapok támogatása *(.AbsolutePage, .AbsolutePosition)*	nem	igen
Könyvjelzők *(.Bookmark)*	nem	igen
Kurzor vissza *(.MovePrevious)*	nem	igen

A különféle paraméterekkel létrehozott Recordset objektumok jellemzői

A Query objektum néhány további jellemzője lehet *(ezeket még a lekérdezés futtatása előtt kell beállítani)*:

- .CodePage – A lekérdezésben, illetve tulajdonságnevekben használt kódtábla azonosítója *(segítséget lásd később)*
- .LocaleID – láttuk, hogy az Indexing Service működésében fontosak a nyelvi beállítások. Itt beállíthatjuk, hogy a lekérdezés milyen lokállal működjön *(de ha nem tesszük, az ASP környezetben futó Query objektum trükkös módon titokban*

kiolvassa a böngésző által – a HTTP_ACCEPT_LANGUAGE fejlécben – küldött értéket)

- .MaxRecords – itt megadhatjuk, hogy a válaszként visszaadott Recordset legfeljebb hány rekordot tartalmazzon

A Recordset visszaadása *(azaz a .CreateRecordset() metódus meghívása)* után a Query objektum bizonyos jellemzői tájékoztatnak a keresés futtatásakor fennálló bizonyos tényezőkről. Konkrétabban:

- .OutOfDate – a jellemző értéke True, ha az adott katalógus elkészítését az Indexing Service még nem fejezte be *(új dokumentumok jöttek létre, meglévők tartalma változott, stb.)*.
- .QueryIncomplete – értéke True, ha a keresést az Indexing Service nem volt képes maradéktalanul kiértékelni és végrehajtani *(például mert a kereső kérdés túl összetettre sikerült)*
- .QueryTimedOut – értéke True, ha a keresés nem fejeződött be az Indexing Service MaxQueryExecutionTime konfigurációs paraméterében meghatározott idő *(alapértelmezésben 10 másodperc)* alatt. Ezt az értéket a Registry-ben, illetve az adminisztrációs objektumok segítségével módosíthatjuk. Bővebb információt az Indexing Service referenciájában találunk.

Egy fontos metódus maradt még a végére: a Query objektum .Reset() metódusa törli az objektum belső állapotát, így az újra felhasználható lesz a további keresésekkor.

A Utility objektum

Az Indexing Service másik, lekérdezéshez használatos objektuma a Utility *(IXSSO.Util)*. Ez az objektum néhány olyan hasznos metódust tartalmaz, amely megkönnyíti a lekérdezések kezelését. Az első, és legfontosabb metódus az .AddScopeToQuery().

```
Set oQuery = Server.CreateObject("IXSSO.Query")
oQuery.Catalog = "Web"
oQuery.Query = sSearch
oQuery.Columns = "FileName, Path, Rank, Vpath"

Set oUtil = Server.CreateObject("IXSSO.Util")
oUtil.AddScopeToQuery oQuery, "c:\indexdemo"
```

Amikor a Query objektumot létrehozzuk, a keresés alapértelmezésben a teljes katalógusra vonatkozik. A katalógusban történő keresést is korlátozhatjuk azonban a katalógus egy vagy több elemére *(itt scope a neve, de tulajdonképpen a könyvtárakról van szó)*. A Utility objektum .AddScopeToQuery metódusával a lekérdezést befolyásolhatjuk. A metódust többször is meghívhatjuk, így egynél több alkönyvtárat is meghatározhatunk a kereséshez. A metódus paraméterei sorra a következők:

- Az első paraméter maga a Query objektum
- A második paraméterben átadhatjuk a kívánt scope nevét *(fizikai vagy virtuális elérési útját)*
- A harmadik paraméter elhagyható; ha megadjuk, értéke „deep" vagy „shallow" lehet. Az előbbi esetén a keresést a megadott scope alkönyvtáraiban is lehetővé tesszük, a „shallow" csak a megadott könyvtár keresését engedélyezi.

Két további metódus az ASP feldolgozást hivatott segíteni. Az ASP Server objektumának .HTMLEncode() és .URLEncode() metódusai már ismerősek lehetnek. Ezek konverziós rutinok, amelyek segítenek a szövegek HTML kódba, illetve URL-be illeszthető formájúvá alakításában. A Utility objektum két hasonló nevű metódust tartalmaz, amelyek az ASP-s változathoz képest kicsit fel lettek okosítva. Mindegyik metódus egy-egy plusz paramétert is hordoz: megadhatjuk *(sőt, meg is kell adnunk)* a konverzióhoz használt kódtábla azonosítóját.

A .TruncateToWhiteSpace() metódus két paramétert vár: az első egy szöveg, a második egy számszerű érték. A metódus a szöveget elvágja úgy, hogy a vágás szóhatárra essék *(azaz nem vág félbe semmit)*, és a visszaadott szöveg legfeljebb a megadott hosszúságú lesz. Ezt a metódust például a szöveges kivonatok *(„characterization" dokumentumjellemző)* méretre vágásához használhatjuk, valahogy így:

```
oUtil.TruncateToWhitespace(
  oRS("Characterization"),200)
```

Végére maradt két konverziós rutin: az .IsoToLocaleID() metódus egy ISO 639 nyelvazonosítót *(pl. „hu")* lokál azonosítóra konvertál *(emlékszünk még a Query objektum .LocaleID jellemzőjére?)*; míg a .LocaleIDToISO pont ennek ellenkezőjét teszi.

```
oQuery.LocaleID = oUtil.IsoToLocaleID(
  Request.QueryString("HTTP_ACCEPT_LANGUAGE"))
```

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://msdn.microsoft.com/library/en-us/indexsrv/indexingservicestartpage_6td1.asp
[2] http://msdn.microsoft.com/library/en-us/indexsrv/ixrefint_9sfm.asp
[3] http://www.adobe.com/support/downloads/detail.jsp?ftpID=1276
[4] http://www.morphologic.hu
[5] http://msdn.microsoft.com/library/en-us/indexsrv/ixqlang_92xx.asp
[6] http://msdn.microsoft.com/library/en-us/indexsrv/ixref_6xid.asp
[7] http://technet.netacademia.net/download/asp/12/
[8] http://msdn.microsoft.com/library/en-us/indexsrv/ixuwebov_1qcn.asp

HTML lite

A Wireless Markup Language

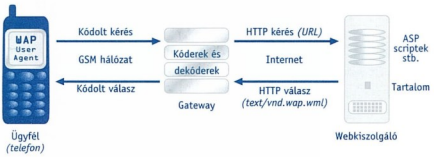
Egy vérbeli informatikus mindenekelőtt hidegrázást kap a csigalassú, legfeljebb kétszintű grafikával *(ha egyáltalán...)* ellátott, 4-5 sorban, soronként 20 karakterben megjelenített szolgáltatásoktól, amit ráadásul rendes billentyűzet hiányában mindenféle trükkös módon lehet csak használni. Kínszenvedés az egész. A WAP halálra van ítélve, mondjuk kórusban. Aztán a kezünkbe kerül az első WAP-os telefon...

Napjainkban egyre elterjedtebbek a WAP-os böngészőkkel ellátott mobiltelefonok. Ezek a kis eszközök a rendelkezésre álló *(gyorsnak nem mondható)* kommunikációs csatornákon keresztül képesek web-szerű szolgáltatások igénybevételére.

Kacérkodunk a gondolattal: itt a lehetőség. A függőség bizony komoly dolog. Elmegyünk pecázni a világ végére, ahol nincs hálózat, nincs Internet, nincs semmi. Térerő azért akad, és lassan rájövünk, hogy csak jó lenne tudni, mi történt a nagyvilágban, hogy megy-e még a szerverünk, kaptunk-e levelet valakitől, mik a lottószámok *(azért a Maldív-szigeteken nem ugyanaz pecázni, mint mondjuk a Tiszán... – sőt, ha bejön, akár a csónakból foglalhatjuk a repülőjegyet :-))*, vagy mondjuk hogy mit jelent a „cucko" magyarul. Végül pedig – hacsak még nem esett a vízbe a csónakról – előkerül a kütyü. A WAP tehát él. Rejtőzködik, aztán lesből támad. Mi pedig áldozatok vagyunk. Addig jó, amíg nincs WAP-os telefonunk :-)

A WAP kommunikáció

A WAP önmaga a telefonkészülék és a szolgáltatónál található WAP-os átjáró *(gateway)* közötti kommunikációt jelenti. A kommunikáció tárgya pedig a mobiltelefonokra optimalizált webes tartalom, amit rendszerint egy, a HTML-hez hasonló, de optimalizált nyelven, a WML-ben írjuk le. A WML egy *(nagyon!)* szigorúan definiált XML formátum. Erre azért van szükség, mert a WAP gateway a kiszolgálótól érkező WML tartalmat tömöríti, tokenizálja, bináris kódra fordítja, és csak ezt a bináris eredményt küldi el a telefonnak. Az automatikus fordításhoz pedig szigorú szintaxisra van szükség *(a dolog finomsága az, hogy az esetleges szintaxishibák a gateway-en derülnek ki, és se a kiszolgáló, se a telefon nem értesül a hiba pontos jellegéről...)*. A WML-t kifejezetten a mobiltelefonok kijelzőjére és a korlátozott adatbeviteli eszközökre optimalizálták.



Egy átlagos WAP-os kommunikáció vázlata

A fenti vázlatból is látszik, hogy a gateway és a kiszolgáló közötti kommunikáció a szabvány HTTP csatornán *(TCP 80 port)*, szabványos formában zajlik *(tehát, egy hagyományos IIS is képes WAP-os szolgáltatásokat nyújtani)*. Az egyetlen dolog, amire figyelni kell, hogy az oldalak MIME típusa („formátuma") ne text/html, hanem text/vnd.wap.wml legyen. Ezt a script kódjából, és külső konfiguráció segítségével is elérhetjük, de ez egy következő cikk témája. Miután a gateway megkapta a kódot, „lefordítja" és továbbítja a készülék felé. Ennyi az egész.

A feladat már csak az, hogy előállítsuk a megfelelő dokumentumokat. A dokumentumok leírásának nyelve pedig a WML. Ugyanolyan könnyű WML oldalakat előállítani, mint HTML-t, csak kicsit más a nyelvjárás.

Wireless Markup Language *(WML)*

A WML tehát mobileszközökre *(és nem feltétlenül csak telefonokra, hanem PDA-kra és egyéb kütyükre – közös jellemzőjük a korlátozott képességű I/O eszközök, és a lassú, esetenként nagy késleltetési idejű hálózati kapcsolat)* tervezett dokumentumleíró-nyelv. Cikkünkben a ma talán legelterjedtebben támogatott WML 1.2 **[1]** verzió alapjait mutatjuk be, de akit a konkrét referencia, a napi változások, illetve a további WAP szabványok népes családja érdekel, bátran látogasson el a W@Pforum.org **[2]** weblapjára.

A WML nyelv elemeit négy fő csoportra oszthatjuk:

- WML struktúraelemek – amelyek magának a WML dokumentumnak a létrehozásához szükségesek
- Szövegmegjelenítés és formázás – néhány, a HTML-ből már ismert karakter – és szövegformázó elem
- Navigáció és hiperlinkek – kapcsolatok, hivatkozások WML dokumentumon belül és oldalak között, természetesen az elmaradhatatlan kérdőívekkel és elemekkel
- Szöveges paraméterek és állapotmenedzsment – a WML nagy előnye, hogy a böngésző képes a WML kódba ágyazott változókat futás közben kiértékelni és feldolgozni

Ne felejtsük el, hogy a WML valójában egy szigorú XML dokumentum, így menet közben be kell tartanunk az XML szintaktikai előírásait. A leggyakoribb hibák *(HTML programozók, figyelem!)*:

- Az XML formátum érzékeny a kis- és nagybetűkre!
- Az elemek attribútumait minden esetben idézőjelek *(")* vagy aposztrófok *(')* közé kell írni

- Mindig minden elemnek van záró tagja. A nyitó tag lehet egyben a záró tag is, ilyenkor az XML-ből ismerős formátumot kell használni *(figyeljük meg a / jelet az elem végén)*, pl:

```
<a href="index.wml">Ennek van záró tagja</a>
```

illetve

```
<br/> vagy <img src="kep.wbmp" alt="kep"/>
```

- XML-ben az elemek nem fedhetik át egymást, tehát az alábbi példa WML-ben is hibás:

```
<u><i></u></i>
```

- Kötött, hogy mely elemek mely más elemeket tartalmazhatnak! Ezért jó, ha mindig kéznél van egy jó kis WML referencia, ha nem is rögtön a specifikáció. A **[3]** címről egy rövid, tömör WML referencia tölthető le, PDF formátumban.

Aki wapos oldalakat fejleszt, hamar rájön, hogy egy jó fejlesztőeszköz nélkül nem fog sokra menni. Hiába a mobiltelefon, a hibákból csak annyit látunk, hogy valami nem fog működni. Hiába a Windows-os wapböngésző, ha az meg közvetlenül dolgozik a WML kóddal. Kellene egy „hamis" gateway is, ami a szemünk láttára fordítja le a kódot, és akkor végre láthatjuk, mi történik majd a mobilszolgáltatónál *(is)*. Egy ilyen, nagyon kellemes, és ingyenes eszköz az Openwave SDK **[4]**, amely többek között egy wapos gatewayt, és a cikk elején látható mobiltelefon-emulációt is tartalmaz. A Nokia fejlesztői fórumának weboldalán **[5]** pedig ingyenes regisztráció után részletes WML referenciákhoz, telefondokumentációkhoz, emulátorokhoz *(pl. a komplett Nokia Mobile Internet Toolkit-hez)* juthatunk. Ilyen segítség nélkül nem érdemes nekikezdeni a munkának.

Fontos! A telefonok memóriája korlátozza a letölthető WML fájlok méretét. Az átlag ma az 1.3 kilobájt *(a Nokia telefonok specifikációját ugyancsak az* [5] *címen lehet megtalálni)*, amit a tömörített, bináris adatfolyamra kell érteni. Az emulátorok általában jelzik, hogy a WML kódból mekkora bináris csomag jött létre. Ennél nagyobb desk-ek letöltése hibát fog okozni, ezért óvakodjunk tőle!

A WML fájl formátuma

A tipikus „Helló Világ" alkalmazás WML-ben így néz ki:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML
 1.1//EN" "http://www.wapforum.org/
 DTD/wml_1.1.xml">
<wml>
<card id="card1" title="WAP teszt">
  <p>Helló Világ!</p>
</card>
</wml>
```

Az első sorban láthatjuk a kötelező XML fejlécet, benne utalással a tartalom kódolásához használt kódtáblára. Itt használhatunk az utf-8-on kívül más kódolást is, ha arra lenne szükségünk. A következő sor *(!DOCTYPE)* a WML szintaxisát definiáló DTD fájlra mutat. Ezután pedig, a `<wml>` elem megjelenésével, az oldal leírása következik. Egy WML oldal egy vagy több kártyát *(card)* tartalmazhat *(emiatt a WML fájlokat az angolban WML deck-nek (kártyacsomagnak) is nevezik)*. A kártya *(amit a `<card>` elem definiál)* egy „oldalnyi" logikailag összetartozó szöveges, formázó, és adatbeviteli elemeket tartalmaz. Egy kártya nem feltétlenül egy telefonképernyőnyi adat, egyszerűen csak egységként kezelendő elemek összessége. A letöltés *(kommunikáció)* egysége viszont mindig a deck, azaz a komplett WML fájl. A linkek, hivatkozások egyaránt mutathatnak egy deck adott card-jára *(ilyenkor a `<card>` elem `"id"` attribútumának értéke egy # jel után belekerül a hivatkozásba – mint HTML-ben a bookmark)*, vagy mutathatnak magára a deck-re is *(ilyenkor a fájlon belül található első kártya fog megjelenni)*.

Lássunk egy példát, ami három kártyát tartalmaz! *(Terjedelmi okokból az XML és DOCTYPE fejlécet ezentúl nem jelezzük, de jegyezzük meg, hogy mindig ott a helyük!)*

```
<wml>
<card id="card1" title="WAP teszt">
  <p>Első oldal.
    <a href="#card2">Tová bb...</a></p>
</card>
<card id="card2" title="WAP teszt">
  <p>Má sodik oldal.
    <a href="#card3">Tová bb...</a></p>
</card>
<card id="card3" title="WAP teszt">
  <p>Harmadik oldal.
    <a href="#card1">Tová bb...</a></p>
</card>
</wml>
```

A bekezdések jelölésére való `<p>` és a hivatkozások létrehozásához használatos `<a>` elemet a HTML-ből ismerhetjük. Nézzük meg az `<a>` elem `href` attribútumát, azaz a hivatkozás célpontját! Az egyes kártyákra a fájlon belül a kártya azonosítójával hivatkozunk, pl.:

```
<a href="#card2">Tová bb...</a>
```

Ha pedig „kívülről", más oldalról szeretnénk egy fájl *(nevezzük mondjuk „cards.wml"-nek)* adott kártyájára mutatni, ezt kell írni:

```
<a href="http://wap.falatrax.hu/cards.wml#card2">
Tová bb...</a>
```

A <head>, <meta> és <access> elemek

A HTML-hez hasonlóan a WML dokumentumnak is lehet fejrésze. A `<head>` elem segítségével, közvetlenül a `<wml>` elemen belül definiált fejrész különféle információkat tartalmazhat az oldalra vonatkozóan. Ezeket a metainformációkat `<head>` elemen belül definiálhatjuk a következő két elem segítségével:

Az `<access>` elem segítségével korlátozhatjuk, hogy egy WML fájl letöltésére honnan érkezhetnek kérések *(azaz, a felhasználó milyen címről juthat el az adott oldalra)*. Ezzel egy bizonyos szinten korlátozhatjuk a wapos oldalainkhoz való hozzáférést. Az `<access>` elem két attribútuma az alábbi lehet:

- `domain`, `path` – mindkét jellemzőnek szöveges értéket adhatunk. A `domain` attribútum értéke a hívó oldal kiszolgálójának tartománynevét, a `path` pedig a hívó oldal elérési útját korlátozza.

A `<meta>` elem különféle egyéb metainformációk definiálását segíti, a fontosabb attribútumok:

- `name` – a tulajdonság neve
- `http-equiv` – mint a `name`, de ha ezt használjuk helyette, akkor a kiszolgáló *(vagy a WAP gateway)* HTTP *(WSP)* fejlé-

cet hoz létre az itt meghatározott adatok alapján
- `content` – a tulajdonság értéke
- `forua` – ha értéke `true`, akkor a beállított adat a böngészőnek szól *(különben nem éri el, mert a WAP gateway feldolgozza és eltávolítja azt)*

```
<wml>
<head>
  <access domain="myorg.hu" path="/mypath"/>
  <meta http-equiv="Cache-Control"
     content="max-age=0"/>
</head>
<card id="cl"><p>Helló Világ!</p></card>
</wml>
```

Szövegformázó elemek

A WML is tartalmaz a HTML-hez hasonló szövegformázó elemeket. Az alapvetőbbeket *(bekezdés, sortörés)* a legtöbb kliens ismeri, de több olyan elem is van, amelyek hatásában nem lehetünk biztosak. Ne feledjük, hogy a WML egyelőre nem a csillogásról szól...
- `<p>...</p>` – Bekezdés határait jelöli. A HTML-lel ellentétben itt mindig minden bekezdésnek vége is van, tehát a záró `</p>` elemet mindig kötelező kirakni. Minden kártya tartalmaz legalább egy `<p>` elemet, mert a `<card>` elem csak ezt az egy szövegformázó elemet tartalmazhatja *(ezen belül aztán már jöhet a többi is)*. A `<p>` elem `align` attribútuma a szöveg balra *(„left")*, jobbra *(„right")* vagy középre *(„center")* rendezését jelezheti, pl.:

```
<p align="center">Helló Világ!</p>
```

A `mode` attribútum értéke *(„wrap" vagy „nowrap")* azt jelzi, hogy a bekezdés tartalma sorokra törtelhető-e, avagy sem *(ha ez lehetséges)*.
- `<br/>` – Sortörés *(egyben nyitó és záró tag is, soha ne felejtsük el kiírni a / jelet a végére!)*.
- `<b>...</b>` – Vastag betűvel megjelenítendő szöveg
- `<i>...</i>` – Dőlt betűs szöveg
- `<u>...</u>` – Aláhúzott szöveg
- `<big>...</big>` – Nagyobb betűméret
- `<small>...</small>` – Kisebb betűméret
- `<em>...</em>`, `<strong>...</strong>` – Általános szövegkiemelés *(hacsak nincs kifejezetten szükségünk egy-egy adott szövegkiemelési típusra (dőlt, aláhúzott, stb.), használjuk helyette ezeket az elemeket)*

Speciális karakterek

A kódban előforduló speciális karaktereket már a HTML-ből is ismerős kódolt formában lehet és kell írni. Az XML szintaxisban fontos szerepet játszó jeleket kötelező így írni, míg a hagyományos karaktereket (pl. ő, ű, stb.) az Unicode-nak vagy más használt kódtáblának köszönhetően nem muszáj. Amit viszont feltétlenül kell:

Jel	Kód	
"	&quot;	&#34;
'	&apos;	&#39;
&	&amp;	&#38;
<	&lt;	&#60;
>	&gt;	&#62;
$	$$	

Gyakran használt jelek és kódjaik WML-ben

Különös figyelmet érdemel a dollárjel. Később még lesz szó róla, de a $ jel WML kódban mindig egy változó nevének kezdetét jelöli, aminek az értékét az oldal megjelenítése közben maga a böngésző értékeli ki. Ezért, ha egyszerű dollárjelet szeretnénk írni, azt a szövegben meg kell kettőznünk. A másik érdekesség az, hogy a & jel még az URL-ekben sem szerepelhet kódolatlanul, tehát az alábbiak közül csak a második sor helyes:

```
<a href="index.wml?a=1&b=2">klikk</a>
<a href="index.wml?a=1&amp;b=2">klikk</a>
```

Táblázatok

Nem tévedés, alapvető formázási célokra használhatunk táblázatokat is. A WML erre a HTML-ből már ismerős táblázat (`<table>`), sor (`<tr>`) és cella (`<td>`) elemeket használja. A `<table>` elem csak `<tr>`-t, ez utóbbi pedig csak `<td>` elemet tartalmazhat, a cella tartalma pedig formázott szöveg lehet *(<p> például nem)*. Az itt látható formációt például az alábbi kódrészlet segítségével hoztuk létre:

```
<table title="Teszt" columns="3" align="RCL">
  <tr>
    <td>--1--</td><td>b</td><td>3</td>
  </tr>
  <tr>
    <td>d</td><td>--5--</td><td>f</td>
  </tr>
  <tr>
    <td>7</td><td>h</td><td>--9--</td>
  </tr>
</table>
```

A példa azt hiszem magáért beszél. A három elem közül csak a `<table>` rendelkezik fontosabb attribútumokkal:
- `columns` – a tábla oszlopainak száma, kötelező megadni. Értéke nem lehet 0.
- `align` – a cellákban található szöveg rendezése, oszloponként. Minden oszlopnak egy betű felel meg, ami lehet „L" *(balra, alapértelmezés)*, „C" *(középre)*, vagy „R" *(jobbra)* rendezés

Navigáció

A legegyszerűbb navigációs elemről, az `<a>`-ról már ejtettünk pár rövid szót az előző oldalakon. A WML azonban jóval összetettebb eseménykezelő és navigációs elemeket is tartalmaz *(még szerencse, mert mire mennénk például kérdőívek nélkül?)*. Mint minden böngésző, a wapböngésző is tárolja bizonyos ideig az addig bejárt oldalak címeit. Ha akarjuk, később ebben a listában navigálhatunk oda-vissza, vagy törölhetjük azt. Az egyszerű navigáció elhelyezi az aktuális oldal címét a history-ban, míg prev művelet például leveszi az utolsó előttit és megnyitja azt. Ebben a tekintetben az egyszerű hiperlinkre „kattintás" is csak egy esemény, ami egy navigációs műveletet *(task)* vált ki.

A műveletek

Négyféle művelet létezik, ezeket egy-egy WML elem személyesíti meg. A művelet további jellemzői értelemszerűen az elemek attribútumaként állíthatók be. Ezek az elemek, az egyszerűtől a bonyolultabbig, az alábbiak lehetnek:

- `<noop/>` – nincs művelet *(bármilyen furcsa is, még ennek is van értelme)*
- `<refresh/>` – az oldal frissítése, újbóli megjelenítése a változók tartalmának újbóli kiértékelése után
- `<prev/>` – visszalépés a historyban
- `<go>` – navigáció és/vagy kérdőív-adatok küldése

Ez utóbbi elemet tárgyaljuk ki egy kicsit részletesebben. A `<go>` elem többek között az alábbi attribútumokat tartalmazhatja:

- `href` – a cél URL-je
- `method` („GET" vagy „POST") – A HTTP kérés típusa, mint a HTML-ben. „GET" esetén az esetleges paramétereket az URL, míg „POST" esetén a HTTP kérés törzse tartalmazza *(ez utóbbit pl. kérdőívek küldésénél kell használni)*
- `sendreferer` *(„true" vagy „false")* – „true" értéke esetén a kérésben benne lesz az aktuális oldal címe

A `<go>` elem egy vagy több `<postfield>` elemet is tartalmazhat. Ezek az elemek definiálják a HTTP kérésbe kódolandó változók nevét és azok értékét *(tehát pl. egy kérdőív által összegyűjtött adatokat)*. Két példa, egyelőre kérdőív nélkül: az első egy „/index.wml?x=1" tartalmú HTTP kérést generál:

```
<go href="index.wml" method="get">
  <postfield name="x" value="1">
</go>
```

A következő pedig v1 és v2 értékét a HTTP kérés törzseként küldi el *(a POST metódusnak köszönhetően)*:

```
<go href="index.wml" method="post">
  <postfield name="v1" value="100">
  <postfield name="v2" value="200">
</go>
```

A műveletek azonban nem járnak csak úgy, egyedül. Ahhoz, hogy a felhasználó kommunikálhasson az oldallal, „parancsokat" kell definiálnunk. Egy-egy parancs a böngészőtől függően megjelenhet grafikus elem *(gomb)*, előre definiált nyomógomb, vagy menüből kiválasztható utasítás formájában, a lényeg, hogy a felhasználó valamilyen módon arra hivatkozhat. A parancs kiválasztása után végrehajtódik a hozzárendelt művelet.

A `<do>` elem

A parancsok általános definiálására a `<do>` elem való. Az elem feladata tulajdonképpen csak az, hogy megértesse a böngészővel, hogy az általa tartalmazott műveletet milyen formában ajánlja fel a felhasználónak. Itt definiálhatjuk például azt, hogy mi lesz az a felirat, amit a böngésző megjelenít. Az elem legfontosabb attribútumai:

- `label` – a parancs neve. Ez az a szöveg, ami megjelenik majd a felhasználó előtt. Próbáljuk minél rövidebbre fogni, az ajánlott hosszúság legfeljebb 6 karakter *(hosszabb is lehet, csak a böngészőn múlik, hogy mit kezd vele)*
- `type` – a parancs típusának definiálására azért lehet szükség, mert előfordulhat, hogy a böngésző *(vagy az eszköz)* előre definiált funkciókat tartalmaz *(pl. az OK gomb mindig egy adott helyen található)*. Ha a böngésző tudja, hogy az éppen definiált parancs tulajdonképpen egy OK *(vagy másik példa: mondjuk egy „HELP")*, akkor megfelelően el tudja azt helyezni. Több előre definiált típus is létezik, de a böngészők bármit elfogadnak, legfeljebb ismeretlenként *(alapértelmezettként)* értelmezik azt, pl.:

Típus	Leírás
accept	Jóváhagyás, elfogadás, OK
prev	Navigáció visszafelé
help	Segítség kérése
reset	Valamilyen állapot törlése, visszatérés alapállapotba
options	Egyéb műveletek, opciók indítása
delete	Elem vagy választás törlése
unknown	Ismeretlen *(alapértelmezett)*

Jegyezzük meg, hogy az attribútum értéke csupán formalitás, a parancs „elhelyezésében" segít az eszköznek, önmagában nem végez műveletet (azaz, pl. egy „prev" típusú `<do>` elemmel nem lehet navigálni, ehhez kell majd egy `<prev/>` művelet is). Ha tehát oldalunkba egy „Vissza" feliratú parancsot szeretnénk elhelyezni, amit kiválasztva a felhasználó a historyban egy oldallal visszalép, a következőt kell írnunk:

```
<do label="Vissza" type="prev">
  <prev/>
</do>
```

Egy sor közben definiált hiperlink (pl.

```
<a href="index.wml">Klikk</a>
```

egyenrangú az alábbiakkal:

```
<do label="Klikk" type="unknown">
  <go href="index.wml"/>
```

A `<do>` elemet a kártyán belül bárhol elhelyezhetjük, de a böngészőn múlik, hogy hogyan jeleníti azt meg. Az `<a>` elemmel meghatározott hivatkozás a szövegben ott jelenik meg, ahol azt definiáltuk.

Események

A `<do>`-hoz hasonló elem az `<onevent>`. Az `<onevent>` elemben definiált műveletet *(`<noop/>`, `<refresh/>`, `<prev/>` vagy `<go>`)* azonban nem a felhasználói beavatkozás, hanem egy bizonyos esemény bekövetkezte kezdeményezi. Négy ilyen esemény létezik, ebből az első hármat card vagy template szinten definiálhatjuk, az utolsót pedig az `<option>` elem tartalmazhatja:

- `ontimer` – az időzítő lejárt *(lásd később a `<timer>` elem ismertetésénél)*
- `onenterforward` – a felhasználó továbblép, például követ egy hivatkozást, vagy egy `<go>` parancs hajtódott végre
- `onenterbackward` – a felhasználó visszalép, például a historyból navigál, vagy egy `<prev/>` parancs hajtódott végre
- `onpick` – ez az esemény egy `<option>` elemben hajtódhat végre, azt jelzi, hogy a felhasználó a szülő `<select>` adatbeviteli mezőben az adott opciót választotta *(ez azt is jelenti egyben, hogy az `<option>` elem tartalmazhát `<onevent>` elemet is)*.

A fenti eseményeket az érintett `<onevent>` elem `type` attribútumaként adhatjuk meg. Ha a böngésző olyan `<onevent>` elemet talál, amelynek típusa nem szabványos, vagy nem illeszkedik a szülőelemhez, az eseménykezelőt figyelmen kívül hagyja.

A `<card>` elem attribútumai

A `<card>` elem fontosabb attribútumai az alábbiak:

- `title` – a kártya címe *(a böngészők általában az oldal tetején, külön sorban jelenítik meg)*
- `newcontext` – ha értéke „true", akkor a böngésző a card megjelenítése előtt törli a history-t és az oldalhoz esetleg tárolt értékeket. Az alapértelmezés „false"

- `ordered` – ha értéke „true", a card-on található beviteli mezők megjelenítési *(és kitöltési)* sorrendje kötött. Akkor használhatjuk, ha olyan kérdőívet töltünk ki, aminek minden mezőjét kötelezően meg kell adni.
- `onenterforward`, `onenterbackward`, `ontimer` – egyszerűsített eseménykezelő rutinok; ha ezeknek az attribútumoknak átadjuk az adott esemény bekövetkeztekor betöltendő oldalak URL-jét, nem kell a kártyán belül definiálnunk a megfelelő `<onevent>` elemeket.

A `<template>` elem

A WML deck a `<card>`-okon kívül még `<template>` elemet is tartalmazhat. Ezt az elemet arra használhatjuk, hogy benne olyan parancsokat, eseményeket definiáljunk, amelyek a deck összes kártyájában érvényesek lesznek *(tehát mintha az összesben egyenként létrehoztuk volna)*. Minden `<do>` elemnek adhatunk saját nevet *(erre való a „name" attribútumuk)*. Ha egy `<card>` egy, a template-ben definiált `<do>`-val megegyező nevű `<do>` elemet tartalmaz, a kártya szintjén definiált parancs felülbírálja a közös definíciót. Ha a kártyában található `<do>` parancs a `<noop/>` műveletet tartalmazza, az adott parancs az adott kártya megjelenítésekor nem lesz elérhető. A következő példában template szinten létrehozunk egy parancsot, amit a card1-ben meghagyunk, a card2-ben felüldefiniálunk, a card3-ban pedig egyszerűen eltüntetünk:

```
<wml>
  <template>
     <do type="options" name="do1" label="klikk">
        <go href="index.wml"/>
     </do>
  </template>
  <card id="card1">
     <p>Itt örököljük az eredetit...</p>
  </card>
  <card id="card2">
     <p>Itt egészen mást csinál</p>
     <do type="options" name="do1" label="klikk">
        <prev/>
     </do>
  </card>
  <card id="card3">
     <p>Itt nincs is ilyen parancs!</p>
     <do type="options" name="do1" label="klikk">
        <noop/>
     </do>
  </card>
</wml>
```

Változók a kódban

A WML kódban szinte bárhol használhatunk változókat, ahol egyébként valamilyen szöveges értéket adhatnánk meg. Ez lehet egy elem tartalma, vagy pl. egy attribútum értéke is. Több olyan WML elem van *(főképp adatbeviteli elemek)*, amelyek az adatokat automatikusan egy-egy változóban tárolják *(a változó neve pedig megegyezik az elem `name` paraméterében megadott névvel)*. Amikor a változó értékét fel akarjuk használni, a neve elé írt $ jellel hivatkozhatunk rá. A legjobb példa erre valamelyik adatbekérésre való kérdőív-elem *(szövegmező, rádiógomb, stb.)*. Amikor ezeknek az elemeknek nevet adunk, egyben létrehozunk egy *(az elem nevével megegyező nevű)* változót is. A kérdőív-elembe írt adatot mindig az adott változó tartalmazza. A kérdőív elküldésekor pedig a `<postfield>` mezőben felhasználjuk az adott változók tartalmát. Lássunk egy példát, ami adatot kér be két szöveges mezőbe, majd azok tartalmát szabványos HTTP POST kéréssel elküldi a kiszolgálónak további feldolgozás érdekében:

```
<wml><card id="card1" title="Teszt">
<p>Vezetéknév:
<input title="Vezetéknév:" type="text" name="n1"/>
<br/>Keresztnév:
<input title="Keresztnév:" type="text" name="n2"/>
</p>
<do type="accept" label="Rendben">
  <go method="post" href="index.wml">
  <postfield name="name1" value="$n1"/>
  <postfield name="name2" value="$n2"/>
  </go>
</do>
</card></wml>
```

Vegyük észre, hogy nincs a HTML-hez hasonló `<form>` elem, a mezőket egyszerűen csak elhelyezzük az oldal kódjában, majd ha kell, a megfelelő `<go>` művelettel küldjük el az adatokat. A változók értékét magában az oldal szöveges kódjában is felhasználhatjuk. A korábban már említett `<refresh/>` művelet mindig a változók új értékével generálja újra az oldal tartalmát:

```
<wml><card id="card1" title="Teszt">
<p> rj ide valamit:
<input title="Valami:" type="text" name="v1"/>
<br/>Ezt írtad:<br/>$v1</p>
<do type="refresh" label="Rendben">
  <refresh/>
</do>
</card></wml>
```

Időzítés – a `<timer>` elem

Minden `<card>` tartalmazhat egy `<timer>` elemet. Ez az elem egy időzítőt indít el, amikor a card megjelenik, majd kiváltja az „ontimer" eseményt, amikor az időzítés lejárt.

```
<card>
  <onevent type="ontimer">
    <go href="next.wml"/>
  </onevent>
  <timer name="t1" value="100"/>
  <p>Hello World!</p>
</card>
```

A fenti példa körülbelül tíz másodperc múlva továbblép a next.wml lapra. A `<timer>` elem főbb attribútumai:

- `name` – a változó neve, amiben az időzítő-számláló található. Értéke folyamatosan csökken. Ezt az attribútumot nem kötelező megadni.
- `value` – az időzítés értéke, kb. 1/10 másodpercben

Kérdőívelemek, adatbeviteli mezők

Ha már az adatbevitelnél tartottunk, lássuk, milyen adatbeviteli elemek találhatók a WML-ben. A legkézenfekvőbb a tiszta szöveges adatok bevitelére való `<input>` elem. A legfontosabb attribútumok:

- `name` – az elem és az értékét tartalmazó változó neve
- `title` – a mező címe *(ezt az attribútumot a böngésző felhasználhatja a beviteli mező megjelenítéséhez, pl. az oldal fejlécében)*
- `type` – „text" (szöveg) vagy „password" *(jelszó)*
- `value` – a mező alapértelmezett, előre megjelenő értéke *(ha a „name" attribútumban meghatározott változó még nem tartalmaz más értéket)*
- `format` – beviteli maszk *(formátumkódok: A = nagyalakú nem szám karakter, a = kisalakú nem szám karakter, N = számjegy, X = nagyalakú karakter vagy szám, x = kisalakú karakter vagy szám, M és m = bármilyen karakter, * = akárhány darab adott karakter (* után valamelyik korábban említett formátumkód állhat, pl. *a = akárhány csupa kisalakú karakter), 1-9 = mint a csillag, csak adott számú jelre érvényes (pl. 2a = két kisbetű), \c = a \ jel után álló karakter megjelenik a beviteli mezőben)*
- `emptyok` – „true" értéke esetén a mezőt kitöltetlenül lehet hagyni (ha nem adjuk meg, alapértelmezés a „false")
- `size` – a mező „szélessége", karakterben
- `maxlength` – a bekérhető adat maximális hosszúsága, karakterben *(ha nem adjuk meg, elvileg végtelen, gyakorlatilag a böngészőn múlik, mennyit fogad el)*

A másik adatbeviteli elem a feleletválasztós típus. WML-ben ezt a `<select>` és „gyermekelemei" valósítják meg. Lássunk először egy egyszerű feleletválasztós példát:

```
<select name="v1" title="Válassz!">
  <option title="s1" value="s1">Első</option>
  <option title="s2" value="s2">Második</option>
  <option title="s3" value="s3">Harmadik</option>
</select>
```

A `<select>` elem főbb attribútumai:

- `name` – az elem és a választott opció értékét tartalmazó változó neve
- `iname` – egy változó neve, ahova a kiválasztott opció sorszáma kerül (0 = nincs kiválasztott érték)
- `title` – a mező címe
- `multiple` – ha értéke „true", akkor egynél több értéket is ki lehet választani *(ha nem adjuk meg, az alapértelmezés „false")*
- `value` – az elem alapértelmezett értéke *(ha a name attribútumban meghatározott változó még nem tartalmaz más értéket)*
- `ivalue` – az elem alapértelmezében kiválasztott opciójának sorszáma *(ha az iname attribútumban meghatározott változó még nem tartalmaz más értéket)*

Amikor több értéket is kiválaszthatunk, az egyes értékek a változókba egymás után, pontosvesszővel kerülnek bele *(és így is kell őket előre megadni, ha szükség lenne rá)*. Mint a példában látható, a `<select>` elem opciókat tartalmaz*(hat)*, minden egyes sort egy-egy `<option>` elem személyesít meg. Az elem tartalma *(a nyitó és záró tagok közötti rész)* jelenik meg a felhasználónak. Az elem fontosabb attribútumai pedig:

- `title` – az opció „neve"
- `value` – az opció értéke
- `onpick` – ha ennek az attribútumnak értéket adunk *(egy URL-t)*, akkor az opció választásakor a böngésző azonnal a megadott címre ugrik

Az elemcsalád harmadik, új tagja az `<optgroup>` elem. Ennek az elemnek mindössze annyi a célja, hogy a könnyebb kiválasztás érdekében csoportosíthassuk a lehetőségeket.

```
<select name="v2" title="Hányadik legyen?">
 <optgroup title="Egytől-Háromig">
   <option title="1" value="1">Első</option>
   <option title="2" value="2">Második</option>
   <option title="3" value="3">Harmadik</option>
 </optgroup>
 <optgroup title="Négytől-Hatig">
   <option title="4" value="4">Negyedik</option>
   <option title="5" value="5"> tödik</option>
   <option title="6" value="6">Hatodik</option>
 </optgroup>
</select>
```

Az `<optgroup>` elem `title` attribútuma határozza meg, hogy mi legyen a csoport neve, de ennek a megjelenítésen kívül más szerepe nincsen. A böngésző eldöntheti, hogy felhasználja-e ezt az információt, vagy sem. Egyes böngészők esetén először a két csoport közül kell választanunk, majd azok tartalmából, mások pedig egyszerűen sorban megjelenítik az összes opciót.

Képek

A wapos eszközök képesek minimális grafikai elemek megjelenítésére is. Ehhez speciális, kétszínű fájlformátumot, a .wbmp-t *(MIME típus: image/vnd.wap.wbmp)* használják, bár az újabb eszközök már képesek a .gif, sőt, animált .gif fájlok megjelenítésére is. A .wbmp formátum létrehozásához több eszköz is található az Interneten, egyet az **[5]** címről letölthető Nokia Mobile Internet Toolkit is tartalmaz. Figyelem! A képek a WML deck-től külön töltődnek le, de a korábban már említett méretkorlátozás *(általában 1.3 kbájt)* a képekre is igaz!

A képek beillesztésére az `<img/>` elemet használhatjuk. Az elem kötelezően megadandó attribútumai pedig az alábbiak:

- `src` – a képfájl URL-je
- `alt` – alternatív szöveg a kép helyett

```
<card id="c1" title="Garfield"><p align="center">
  <img src="garf75.wbmp" alt="Lusta macska"/>
</p></card
```

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:

[1] http://www.wapforum.org/what/technical_1_2.htm
[2] http://www.wapforum.org
[3] http://www.ayg.com/images/wml_ref.pdf
[4] http://developer.openwave.com/
[5] http://www.forum.nokia.com/

.NET Akadémia
(I. rész)

Új év, új sorozat. Elérkezett az idő itt Magyarországon is, hogy belevágjunk egy ki tudja milyen hosszú ideig tartó kalandba, és feltérképezzük a .NET szövevényes világát. A .NET-ről, mint fejlesztési háttérről már írtam egy bevezetőt a Tech.net 2001. júniusi számban. Aki még egyáltalán nem találkozott a .NET-el, annak ajánlom a cikket elolvasásra, mert az abban leírtak ismeretét már alapul veszem ebben a részben.

.NET alapok

Kezdetben valák a sík API-k. Egy nagy, ömlesztett massza, egy nagy függvénykészlet, amely segítségével kiaknázhattuk a Windows szolgáltatásait. Bajlódtunk kezelőszámokkal *(handler)*, bonyolult struktúrákkal. Furmányos neveket adtak a Windows API függvényeknek, hogy egyértelmű legyen miről beszélgetünk, szó nem volt névterekről vagy bármi egyéb kategorizálásról.

Az MFC, a VB Runtime, és minden nyelv saját könyvtárai próbálták elfedni ezt az alacsony szintű API-t, és lehetőleg objektumorientált módon összefogni a funkcionalitásokat. De a háttérben mindig ott lapult a Windows API.

Nincs ez másként a .NET-ben sem. Ha megnézzük a Windows-ra írt .NET keretrendszer osztálykönyvtárát, akkor a legvégső operációs rendszer szolgáltatások igénybevételekor ott lapulnak a jó öreg API függvények. Azonban igen jelentős különbségek vannak például a VB6 Runtime és a .NET mögötti osztálykönyvtárban. A legjelentősebb különbség oka az a tény, hogy a .NET-es programokat, így az osztálykönyvtár szolgáltatásait is már nem közvetlenül az operációs rendszer futtatja, hanem a .NET saját futtató rendszere. A .NET futtató rendszer viszont csak egyféle formátumú programokat képes futtatni, amelyek Microsoft Intermediate Language *(MSIL)* nyelvre vannak lefordítva. Azaz nem platformfüggő, például x86-os processzorra lefordított kódot futtat a .NET futtató, hanem IL-t. Az IL-t nem folyamatosan értelmezi és futtatja, mint hajdanán a Basic interpreter, vagy a Java virtuális gép, hanem igény esetén, egy-egy metódus meghívásakor lefordítja a metódus törzsét a platformnak megfelelő gépikódra, és azt futtatja a processzor. Tehát végül is gépikódot futtatunk, csak a fordítás IL-ről gépikódra röptében történik meg, egy-egy függvény meghívásakor. Ennek a látszólag időpazarló, lassú eljárásnak számtalan előnye lesz, de talán a legfontosabb, hogy a futtató rendszer pontosan tudja mit szándékozik tenni egy program. Nemcsak reménykedünk abban, hogy egy letöltött .NET komponens nem kezd-e el matatni a saját dokumentumaink között, hanem teljes bizonyossággal tudhatjuk, hogy akar-e ilyet tenni *(és persze előírhatjuk, hogy mit tehet és mit nem)*. Gépikódú programok esetén nagyon nehéz lenne a futtatónak megtudni, pontosan mit is akar egy program tenni, mert a gépikódra kioptimalizált kódban elmosódik a programozó eredeti szándéka. A .NET-es fordítók által generált IL szándékosan olyan szószátyár, hogy a futtató rendszer egzaktul tudja szabályozni, hogy milyen erőforrásokat enged a kód közelébe, és melyeket zár el tőle, és ezt be is tudja tartatni.

Pont az ellenőrzöttsége miatt hívjuk a .NET-es fordítók által generált IL kódot **menedzselt** kódnak. S ki a menedzser? Nos, ő a **Common Language Runtime**, akit barátságosan csak CLR-ként szoktunk emlegetni. ő fogja osztani az összes áldást, amiről a cikksorozat szólni fog.

A .NET mindenre jó?

Sokan azt gondolják, hogy a .NET miatt megszűnik a hagyományos programozás létjogosultsága, azaz senki **nem** fog már nem menedzselt kódot írni *(persze beszéljünk csak a Windows programozókról)*. És mi lesz az eszközmeghajtók íróival? Mi lesz azokkal, akik nagyon gyors, alacsony szintű, kis tárigényű komponenseket akarnak írni? ők nem fogják egyhamar kidobni például az ATL-t *(Active Template Library)*. Aki szeretne generikusan, template-ek segítségével programozni, vagy nem tud meglenni többszörös öröklődés nélkül, az jobb, ha marad a nem menedzselt világban. Mindig is lesznek olyan feladatok, amelyek nem valósíthatók meg menedzselt kódban. Egyre több mindenre lesz jó, de például magát a .NET futtató rendszert is meg kellett írni valakinek, és ugyebár azt nehéz lett volna .NET-ben megírni.

A CLR a benne futó programok számára maga a Mátrix. Egy szimulált világ, ahol ízletesek az ételek, az emberek jól néznek ki, de vannak kötöttségek, számítógépek határozzák meg mit tehetünk és mit nem. Cserébe kapunk egy nagycsomó szolgáltatást, így sokkal gyorsabban meg tudunk valósítani sokféle funkcionalitást. A másik, a való világban szinte mindent megtehetünk *(na, jó, a fizika még működik)*, nem befolyásolnak a gépek, szabadok vagyunk. Cserébe viszont egy sokkal kellemetlenebb környezetben kell programoznunk. Ez a jelen, azaz a nem menedzselt kód. Biztos lesz sok olyan macsó programozó, aki nem hajlandó magát alávetni a CLR rendőrállam intézkedéseinek, és ő soha az életben nem fog menedzselt kódot írni, mert ő nem szereti, ha megkötik a kezét. Tényleg? Akkor ideje visszatérni a DOS-os világba, ott tényleg nem kötötték meg a kezünket. Ha akartuk, közvetlenül belenyúlhattunk az összes hardverjellemzőbe. Ha úgy tetszett akár szektorszinten írhattunk a merevlemezre. Senki nem szólt, hogy hóha, ez a partíciós tábla.

Aki valamilyen mai operációs rendszer alá dolgozik, annak el kell fogadni, hogy meg van kötve a keze. Azért, mert nem csak az ő programja fut egyedül a számítógépen, így védeni kell egymástól és sokszor önmaguktól is a programokat. Erre való az operációs rendszer, és aki nemmenedzselt kódot ír, az is szembeütközik ezzel. Aki tényleg macsó akar lenni, az írjon eszközmeghajtót kernelmódban.

Amit látnunk kell, hogy a .NET egy újabb absztrakciós réteg a fizikai vas, a számítógép hardvere, és a programjaink között. Az oprendszer az első réteg, arra építették fel a CLR-t, ami egy újabb réteg. Az indirekció ára nyilvánvalóan némi teljesítménycsökkenés, cserébe a futtató rendszer jobban kordában tudja tartani a

programokat, ami például megbízhatatlan, internetről letöltött vezérlők és programok (vírusok) esetén rendkívül fontos dolog.
És még van egy óriási előnye. Mi akadályoz meg valakit abban, hogy megírja a CLR-t más operációs rendszerre? Ha a CLR valóban annyira el tudja szigetelni a programunkat a hardvertől, akkor semmi akadálya, hogy az IL formátumú programunk más operációs rendszer alatt, sőt, akár más architektúrán fusson. Csak egy feltétele van ennek: a CLR minden szolgáltatása meg legyen írva, amit a programunk igényel a futtatandó platformon. A Microsoft már jó ideje írja a Linux-os CLR-t...

Csapjunk bele!

Ennyi bevezető után térjünk rá konkrét témákra. Fejleszteni szeretnék .NET-es, menedzselt programokat. Mire lesz szükségem?
A legfontosabb alap a .NET Framework SDK. Ezt egy ingyenesen letölthető telepítőprogram **[1]** rakja fel. Ezután rögtön elindulhatunk a .NET felfedezésére. Első körben nem érdemes felrakni a Visual Studio.NET-et, jobb látni, mi van ténylegesen a .NET mögött. A VS sokmindent trükkösen elfed illetve levarázsol helyettünk, ami nagyon jól jön majd akkor, amikor már értjük, miért pont azt a kódot varázsolta be, amit. De addig is inkább használjunk parancssori eszközöket és egy számunkra kényelmes szövegszerkesztőt. Akár notepad is lehet, bár nekem a fapad nem kényelmes.
A Framework SDK telepítése után a .NET rendszerállományai javarészt a C:\WINNT\Microsoft.NET\Framework\v1.0.2914 mappába kerülnek bele. A v1.0.2914 kiadásonként változik, ez a Beta2 verziószáma *(a cikk megjelenésekor már lesz végleges termék is)*.
Ebben a könyvárban van egy csc.exe, vbc.exe és jsc.exe. Ezek a C#, VB.NET és JScript.NET fordítók. Mivel ezeket és a többi segédprogramot is gyakran fogjuk használni, érdemes a mappát belerakni az operációs rendszer elérési útjába, így bármilyen könyvtárból könnyen futtathatjuk őket.
A cikksorozatban szinte mindig C# nyelven fogom közölni a példaprogramokat. Ahol érdekes, ott megemlítem a VB.NET-es változatot is, de általában a kulcsszavak kicserélésével könnyen átírhatjuk egyik nyelvről a másikra a programokat.
Kezdjük a szokásos Helló világ program megírásával.

```
class ElsoDotnetesProgi
{
   public static void Main()
   {
      System.Console.WriteLine("Helló világ! (1)");
   }
}
```

Írjuk be egy szövegfájlba és mentsük el hello1.cs néven. Nyissunk egy konzolablakot, és fordítsuk le a programunkat!

```
csc hello1.cs
```

A csc.exe a C# fordító, ami bemenetül egy C# forráskódot tartalmazó fájlt vár, és készít egy ugyanolyan nevű .exe-t.
Futtassuk le a hello1.exe-t!

Ez igen! Járjuk körbe a forráskódot!
Ami elsőre szembetűnő, hogy van más nyelvekből megszokott main függvény, ahol a program futása kezdődik, csakhogy ez egy osztály metódusaként van deklarálva. Ennek oka, hogy a .NET-ben nincsenek globális függvények, márpedig a main egy ilyen függvény szokott lenni. A .NET-ben majd osztályokat *(class)* és struktúrákat *(struct)* fogunk írni, és azok metódusait - de sohasem globális függvényeket.
Különösen C programozóknak furcsa lehet, hogy a System.Console.WriteLine metódus használatához nem kellett semmilyen header fájlt beinklúdolni a forrásba, és fordításkor sem kellett megadni annak a kódkönyvtárnak a nevét, ahonnan a metódus kódját kiszedné a fordító. Nos, a fenti metódus egy központi helyre beregisztrált komponens metódusát hívja meg, amelyet a név alapján magától megtalál a fordító. Saját kódkönyvtárainknál segíteni kell neki, de erről majd még értekezünk a komponensek tárgyalásakor.
Az IL-re lefordult hello1.exe egy modul lett. A modul az egy alkalmazás vagy egy kódkönyvtár *(általában .exe és .dll kiterjesztéssel)*. Egy assembly egy vagy több modul összessége. Egy assembly minden modulja tudja, hogy kik a társai, ez a modulok elején található manifest információs blokkban tárolódik. Az assembly lesz majd a verziózás és a telepítés egysége, másképpen fogalmazva hiába áll egy assembly fizikailag több modulból, logikailag egy egységnek tekintendők. Felfogható egyfajta logikai dll-nek is. Azaz a hello1 egy olyan assembly, ami egy modulból áll, és ez a hello1.exe.
A .NET futtató a Main metódust hívja meg a programunk elindításához. Public, azaz nyilvános az ő láthatósági jelzése, így a futtató el tudja érni. Alapban, C#-ban minden osztálylakó private, azaz saját lenne, így még a futtató sem tudná meghívni a Main-ünket. A static kulcsszó pedig azt jelenti, hogy az osztályunkból (ElsoDotnetesProgi) nem kell létrehozni példányt a metódus meghívásához, közvetlenül is megtehetjük azt.
Figyeljük meg, hogy a Main nagybetűvel van írva. A C# kis-nagybetű érzékeny, ami nagy érvágás tud lenni a VB programozóknak, de aki ezt nem tudja megemészteni, még mindig ott a VB.NET.
Az érdemi kiíratást a WriteLine metódus végzi, ami egy statikus metódusa a Console osztálynak, ami pedig a System névtérben lakik. A .NET-ben az osztályokat névterekbe rendezhetjük, így egy-egy funkcionalitást megvalósító, logikailag kapcsolódó osztályokat összefoghatjuk. Például a System.IO névtérben van az összes fájlkezeléssel kapcsolatos osztály. Vagy a System.XML-ben az XML dokumentumok kezelését végző osztályok. A System osztály egy vegyesfelvágott, ebben vannak deklarálva az alaptípusok (int, string, ...), és sok, a típusokon műveletet végző osztály. Nem Helló világ méretű programok esetén nagyon hasznos a saját osztályainkat is megfelelő, leíró jellegű névterekbe terelgetni. Tipikus a Cégnév.Funkció.Alfunkció névtérhasználat, például NetAcademia.Web.Pages.
Habár a System.Console.WriteLine még nem túl hosszú ahhoz, hogy begépeljük, azért a System.Runtime.InteropServices.CustomMarshalers.EnumerableToDispatchMarshaler.MarshalManagedToNative ötödik begépelésekor lehet, hogy a pokolba kívánnánk a névtér kitalálóit. A gépelések csökkentésére a használni kívánt névtereket előre megjelölhetjük a programunk elején, így elég csak az osztályok neveire hivatkozni. Erre a using kulcsszó való (VB: Imports).

```
using System;
class ElsoDotnetesProgi
{
   public static void Main()
```

```
{
   Console.WriteLine("Helló világ! (2)");
   }
}
```

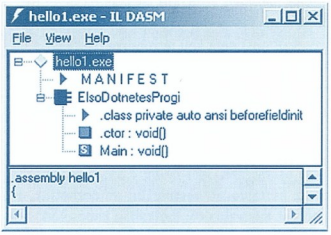
A fordító végignézi az összes, using-okban megadott névteret, amikor egy osztályt keres, a példánkban a System névtérben meg fogja megtalálni a Console osztályt.
Általában nem hivatkozunk az osztályokra a teljes nevükkel, hanem minden használt névtérre felveszünk egy álnevet a using segítségével. Fontos, hogy értsük, hogy az esetleg felesleges using-ok miatt nem lesz nagyobb a lefordult program, ez csak egy egyszerű gépelési segítség nekünk.

A színfalak mögött

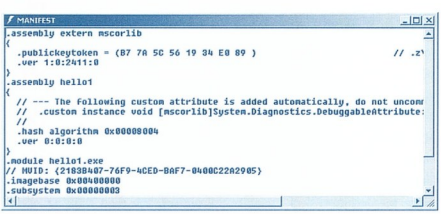
Nézzük meg mivé fordult le a hello1.cs! Lett belőle egy 3 kBájtos .exe. Ez nem rossz. Hogyan nézhetnénk bele? Van egy ildasm.exe nevű IL visszafejtő programunk, szintén települ az SDK-val. Ez a C:\Program Files\Microsoft.Net\FrameworkSDK\Bin mappában rejtőzködik, újabb könyvtár aminek a path-ban a helye.
Nyíljon hát ki a nagyvilág!

```
ildasm hello1.exe
```

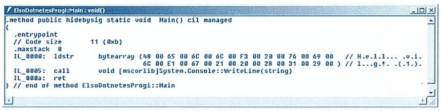
Előugrik az ILdasm, benne a hello1.exe-nk.



Láthatjuk a saját osztályunkat, annak konstruktorát *(.ctor, attól, hogy nem írtunk még generálódott)*, és a Main metódusunkat. A MANIFEST a már említett bevezető információ, ami a hello1 assembly-t írja le:



Ami nagyon érdekes, hogy rögtön az első sorból látszik, hogy a programunk működéséhez szükség lesz az mscorlib nevű assembly-re, annak is a megfelelő verziójú és nyilvános kulccsal hitelesített változatára. A futtató már a manifest felolvasásával le tudja ellenőrizni, hogy egyáltalán van-e értelme elkezdeni futtatni az alkalmazásunkat, minden kódkönyvtár elérhető-e? Ez nem volt jellemző a DLL és COM világban, nem lehetett megmondani, hogy egy programnak pontosan milyen külső DLL-ekre vagy COM komponensekre van szüksége.
A Main metódusra kettőt kattintva felugrik a metódusunk IL-re lefordított kódjának visszafejtett változata. Az első utasítás lenyomja a verembe a kiírandó szöveget tartalmazó bájttömböt *(Unicode string)*, míg a második meghívja a kiíratót függvényt.



Családfakutatás

Az ILDasm-ben kattintsunk kettőt a háromszöggel jelzett .class private auto ansi beforefieldinit feliratra.

```
.class private auto ansi beforefieldinit
ElsoDotnetesProgi extends [mscorlib]System.Object
{} // end of class ElsoDotnetesProgi
```

Látható, hogy az ElsoDotnetesProgi osztályunk az mscorlib-ben lakó System.Object osztályból öröklődik. Hogy lehet ez, hisz nem jelöltünk ki semmilyen öröklődést?
Egy nagyon alapvető és fontos tényre bukkantunk rá: .NET-ben minden osztály és típus implicit módon a System.Object-ből öröklődik. Mind a saját osztályaink és struktúráink, mind a .NET Class Library összes osztálya. Még az olyan primitív típus, mint az integer is az Object-ből származik. Ennek mélyenszántó következményei lesznek, amelyekkel a következő részben, a .NET típusok ismertetésénél részletesen fogunk foglalkozni. Egyenlőre elég annyi, hogy a közös ős miatt minden típus elérhető lesz System.Object referencián keresztül, és egységesen kezelhető lesz mint egy általános Object osztály, másrészt minden objektum támogatni fog néhány olyan alapvető metódust, amit a System.Object-ből örökölt.

Osztály vigyázz!

A .NET leggyakoribb típusa a class, amit magyarul osztálynak szoktunk fordítani. Ismerkedjünk meg vele!
Egy osztály nem más, mint egy sablon objektumok létrehozásához, amely tartalmazza az objektum adatainak leírását és a rajtuk elvégzendő műveleteket is. Az osztályok egyik legfontosabb feladata elfedni a mögöttük rejlő adatok fizikai szerkezetét, és így kívülről számunkra csak az objektumon végezhető műveletek a fontosak és láthatóak. Például legyen egy olyan osztályunk, ami számokat képes tárolni *(kollekció)*. Hogy ez belül egy láncolt listával vagy egy tömbbel van megvalósítva nem tartozik az osztályt használókra. Természetesen a sebesség és karbantarthatóság szempontjából egyáltalán nem mindegy a háttérstruktúra, de ezt ledokumentáljuk a felhasználóknak, hogy milyen jellegű igénybevételre melyik módon megvalósított kollekciónkat ajánljuk neki. Kívülről nem szabad látszania a tényleges tárolási struktúrának, csak biztosítani kell a működéshez szükséges műveleteket. Hozzáadni új elemet a listához *(pl. elejéhez, végéhez, adott pozícióra)*, kitörölni egyet, lekérni vagy beállítani egy elem értékét, satöbbi. Ezek mind az objektumon elvégezhető műveletek voltak, amelyeket a háttéradatoktól függő módon, de mindkét esetben meg lehet valósítani.

```
class JorgomBorger
{
   //tagok
}
```

A kapcsos zárójelek között fogjuk leírni az osztály által tartalmazott adatokat és az osztályon értelmezett műveleteket (tagokat). Hogyan lesz egy osztályból egy konkrét, élő objektum? A new operátor segítségével.

```
JorgomBorger j = new JorgomBorger();
```

A sor végrehajtása után létrejön az osztályunk mintájára egy objektum a szemétgyűjtő által karbantartott halmon *(heap)*. A j egy objektumreferencia, amin keresztül fogjuk tudni elérni az osztály tagjait. Az osztálynév utáni zárójel kötelező C#-ban.
Az objektumorientált irodalomban egy osztálynak kétféle tagja van: adat-tagok vagy mezők *(fields)* és függvény-tagok vagy metódusok *(methods)*. A .NET-ben még két tagot hozhatunk létre: jellemzőket *(property)* és eseményeket *(events)*. Az utóbbi kettő eléggé egyedi a .NET-ben, más rendszerekben interfészekkel és metódusokkal szokták szimulálni őket. Mivel a .NET natívan támogatja őket, elég sok unalmas programozási munkától kímél meg minket.

Mezők

A mezők tárolják az osztály által reprezentált adatokat, másképpen szólva ezek tárolják az osztály állapotinformációit. Kétféle mezőt hozhatunk létre egy osztályon belül. Az egyik minden egyes, az osztályból létrehozott objektumpéldányban egyedileg, egymástól függetlenül létrejön, ezek a példányonkénti mezők *(instance fields)*. Ez az alapértelmezett típus. A másik típus közös minden objektumpéldányra, azaz osztályonként mindig csak egy van belőle. Ezek a statikus mezők *(static fields)*.

```
class JorgomBorger
{
  int a;   //példányonkénti
  static double b;   //közös
}
```

VB-ben a statikus mezőt a Shared kulcsszóval azonosítjuk, ami jobban kifejezi a működését, de az objektumorientált irodalom a static kulcsszót favorizálja.
A mezők tartalma közvetlenül nem tartozik a külvilágra! Ez az osztály belső működéséhez kell, kifelé ellenőrzött módon, metódusokon és jellemzőkön keresztül engedjük csak elérni őket. Ez nagyon fontos alapelv, mert így ki tudjuk cserélni az osztály teljes belső adatszerkezetét anélkül, hogy ebből a külvilág (az osztályfelhasználó programok) bármit is észrevennének. Ez a karbantarthatóság és a bővíthetőség miatt nagyon fontos OOP alapelv.

Metódusok

A metódusok az osztályon végrehajtható műveleteket írják le. A mezőkhöz hasonlóan ezek is lehetnek példányonkénti vagy közösek. A példányonkénti metódusok mindig egy bizonyos példányon dolgoznak.

```
using System;
class ElsoDotnetesProgi
{
  public static void Main()
  {
    JorgomBorger jb1 = new JorgomBorger();
    JorgomBorger jb2;
    jb2 = new JorgomBorger();
    jb2.LegyenAzA(8);

    Console.WriteLine("Ez egyik a: {0}",
                      jb1.NaMiAzA());
    Console.WriteLine("A másik a: {0}",
                      jb2.NaMiAzA());
    Console.WriteLine("          b: {0}",
                      JorgomBorger.NaMiAB());
  }
}

class JorgomBorger
{
  int a = 4;  //példányonkénti mező
  static double b = 5.57;  //közös mező

  //Példányonkénti metódusok
  public int NaMiAzA()
  {
    return a;
  }
  public void LegyenAzA(int x)
  {
    a = x;
  }
  //Egy statikus (közös) metódus
  public static double NaMiAB()
  {
    return b;
  }
}
```

Kimenet:

```
Ez egyik a: 4
A másik a: 8
          b: 5,57
```

Létrehoztunk két JorgomBorger objektumot. A másodiknak meghívtuk a LegyenAzA() metódusát, paraméterül 8-at átadva. A metódus beállítja a példányonkénti „a" változó tartalmát a paraméterül kapott értékre. A NaMiAzA() meghívásával kiíratjuk mindkét példány „a" nevű mezőjének értékét. Láthatjuk, hogy a két „a" különböző, objektumpéldányonként függetlenül léteznek.
Ha létrehozunk egymillió osztálypéldányt, akkor a példányszintű metódusokat alkotó kódok egymilliószor fognak létrejönni a memóriában? Ha így lenne, akkor ki sem alakult volna az objektumorientált programozás. Természetesen csak a példányonkénti mezők tárolásához szükséges memóriaterületet kell lefoglalni egymilliószor, a metódusok kódjai csak egyszer jönnek létre a memóriában. Viszont, hogy egy ilyen „levegőben lógó" metódus tudja, hogy mely adatokkal kell neki foglalkozni, kap egy referenciát (pointert) egy tényleges osztály adataira. Ezt nem látjuk, de a fordító minden példányonkénti metódusnak átadja ezt paraméterül a sajátjaink mellett.
Egy metódusban elérhetjük ezt a saját osztályra mutató referenciát a this kulcsszóval. Például ha az előbbi példában a LegyenAzA() metódus paraméterét „a"-nak hívjuk, mint az osztály mezőjét, akkor a this referenciával explicit jeleznünk kell, hogy ennek az osztálynak az „a" mezőjének adunk értéket.

```
public void LegyenAzA(int a)
{
  this.a = a;
}
```

A statikus metódusoknak nincs szükségük a this referenciára, ezért a hívásuk szintakszisa is más mint a példányonkénti társaiké, jelezve, hogy az osztályra, és nem az egyes példányokra vonatkoznak.

```
JorgomBorger.NaMiAB()
```

C++ programozók talán furcsán néznek az előző sorra, miért nem :: (kettőspontok) választják el az osztály nevét a metódus nevétől? C#-ban minden esetben pontot kell használni, a C++-beli -> helyett is. Bár a .NET-ben nincsenek globális függvények és változók, statikus mezőkkel és statikus metódusokkal valami hasonlót érhetünk el, csak mindig ki kell írni az osztály nevét, illetve be kell csomagolni egy osztályba a függvényeket és a változókat.
Mind a metódusok, mind a mezők alapértelmezésben nem elérhetőek az osztály felhasználó programok számára. Ez is az adatok egységbezárását, elrejtését kívánja elősegíteni. A nagyvilág számára is érdekes metódusokat *(esetleg mezőket)* a public kulcsszóval tudjuk láthatóvá tenni. Az összes többi csak az osztály metódusai számára látható, kívülről nem érhető el, erre figyelmezet a fordító. Ez a private elérhetőség. További elérhetőségi kulcsszavak is vannak.
A protected kulcsszóval megjelölt tagok kívülről nem látszanak, de az osztályon belülről és az osztályból leszármazott osztály metódusai elérhetik őket. Az internal kulcsszóval jelölt tagok pedig csak ugyanabból az assembly-ből érhetők el.

Zárszó

A következő részben megnézzük az osztályok további jellemzőit, valamint áttekintjük a .NET további alaptípusait és a rajtuk értelmezhető műveleteket. Egyet már részben láttunk ebben a részben is: az osztályt. Legközelebb sorra kerülnek a primitív típusok, a System.Object részletesen, valamint az érték és referencia típusok pontos megértése és a közöttük lehetséges konverzió, a boxing megismerése.

Soczó Zsolt MCSE, MCSD, MCDBA
Zsolt.Soczo@netacademia.NET

A cikkben szereplő URL-ek:
[1]: .NET Framework SDK
http://msdn.microsoft.com/downloads/default.asp?url=/downloads/sample.asp?url=/msdn-files/027/000/976/msdncompositedoc.xml
[2]: Letölthető példakódok
http://technet.netacademia.net/download/xml

XMLgessünk
(IX. rész)

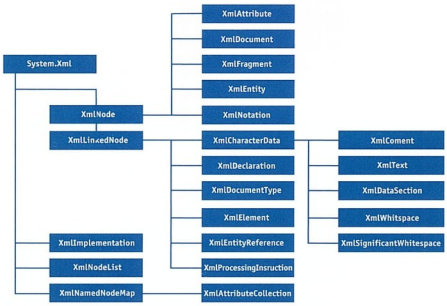
Az előző részben láttuk, hogyan kell használni az XmlReader és az XmlWriter osztályokat. Gyorsak, de eléggé alacsonyszintű interfészeket adnak a kezünkbe. Most megnézzük azokat a technológiákat, amelyek rájuk épülnek, és a segítségükkel így jóval fejlettebb módon is tudunk xml dokumentumokat manipulálni. A főszereplők a DOM, az XSLT és az XPath lesznek.

DOM.NET

A legelterjedtebben használt xml feldolgozó megközelítés a World Wide Web konzorcium által ajánlott XML Document Object Model *(DOM)*. A DOM nem más, mint xml dokumentumok leképezése egy memóriabeli fára, amelyen keresztül bejárhatjuk és módosíthatjuk a dokumentumok tartalmát, majd visszamenthetjük a változtatásokat. A memóriabeli leképezés előny a dokumentum tetszőleges bejárása és módosítása során, viszont hátrány a dokumentum méretével arányos memória felhasználás miatt.

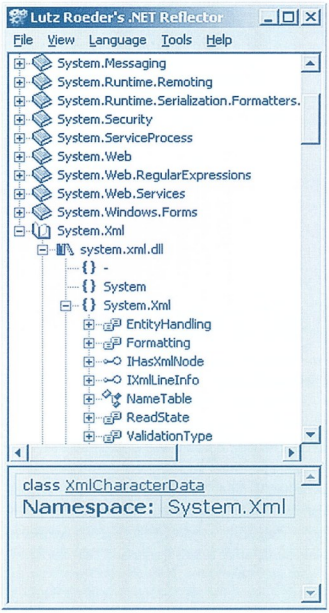
A .NET osztálykönyvtárban az XML DOM-ot a System.Xml.XmlDocument osztályban valósították meg. Dokumentumok betöltésekor az XmlReader-re épít, elmentéskor pedig az XmlWriter-re. Mivel az előző részben láttuk, hogy mindkét osztály csak egy váz *(absztrakt alaposztály)* konkrét író és olvasó osztályok megvalósításához, ezért az XmlDocument képes bármilyen XmlReader vagy XmlWriter megvalósítás adatainak kezelésére, ami - mint látni fogjuk - igen érdekes és hasznos együttműködésre ad lehetőséget egyéb adatforrásokkal: például relációs adatbázisokkal.

A .NET-es DOM osztályhierarchia az alábbi ábrán látható.



A .NET-es DOM osztályhierarchia

Honnan lehet ezt az információt kinyerni *(a dokumentáción kívül)*? Az internetről letölthető egy hasznos kis program, aminek .NET Reflector **[1]** a neve. Nagyon hasonlít a Visual Studio.NET Object Browser-éhez, csak ingyen van, okosabb, és nem foglal el annyi memóriát. A segítségével megnézhetjük, hogy pontosan mi is található egy-egy assembly belsejében, azaz milyen típusok, azokon belül milyen metódusok, mezők, interfészek, satöbbi vannak implementálva. A private tagok is látszanak, mivel az ő leírásaik is benne vannak az assembly-kben *(metaadatok formájában)*, és program által használt Reflection nevű technológián keresztül könnyedén láthatóak.



A .NET Reflector minden részletet megmutat az assemblykből

Könyvvel vannak jelölve az assembly-k, és azokat kibontva láthatóak az őket tartalmazó fizikai fájlok, a bennük lakó névterek, és a névtereken belül a típusok.

A következő képen nézzük meg az XmlNode környékét! A SubTypes ág alatt jól láthatóak az XmlNode leszármazottjai, illetve az XmlLinkedNode gyermekei. Így állt össze a diagram.

A Reflector egy igen hasznos eszköz a .NET felderítésekor, mert valóban azt látjuk, ami a gépünkön van, nem pedig azt, amit a dokumentáció ír. Ez nem azt jelenti, hogy rossz a dokumentáció, csak azt, hogy így béta fázisban még sok benne a kitöltetlen oldal. Egy-egy metódusra kattintva ráadásul a program visszafejti a metódus IL kódját, így akár bele is nézhetünk, hogyan írt meg az MS egy-egy funkcionalitást.

A .NET Reflector egészen a metódusok forrásáig lehetővé teszi az információbogarászást

XmlDocument

A .NET-es XmlDocument nagyon hasonló a MSXML2.DOMDocument COM osztályhoz. A Load() metódus segítségével tölthetünk be egy xml dokumentumot a memóriába. Négyféle felüldefiniált változata is van a metódusnak, a legegyszerűbb egy fájlnevet vagy URL-t vár forrásként. Nem jól formázott vagy elérhetetlen xml forrás esetén XmlException-t kapunk, amit le kell kezelnünk *(try-catch)*.

```
// XmlDocument1.cs

using System;
using System.Xml;
public class XMLDOMTeszt
{
   public static void Main()
   {
      try
      {
         XmlDocument d = new XmlDocument();
         d.Load("a.xml");
      }
      catch(XmlException ex)
      {
         Console.WriteLine(
            "Gáz van a {0}. sorban, " +
            "a(z) {1}. karakternél.",
            ex.LineNumber, ex.LinePosition);
         Console.WriteLine("A hiba oka:\n{0}",
                           ex.Message);
      }
   }
}
```

A memóriabeli dokumentum bejárásához és módosításához megvannak a DOM szabványban definiált szokásos metódusok. CreateElement, CreateAttribute, CreateComment, CreateProcessingInstruction, és társaik egy-egy megfelelő XmlNode objektum létrehozására. Az így létrehozott XmlNode objektumot aztán a kívánt faághoz hozzácsatolhatjuk az AppendChild, InsertAfter vagy InsertBefore metódussal. A RemoveAll, RemoveChild duóval a dokumentumfa egy részét tüntethetjük el, a ReplaceChild-del pedig lecserélhetjük egy ágát.

Az előbbi módosító műveletek elvégzéséhez oda kell navigálnunk a megfelelő node-hoz. Rengeteg bejárási módszer létezik. Az egyszerűbbek a FirstChild, LastChild, ChildNodes és ParentNode jellemzőkön keresztül gyalogolnak a dokumentumban fel-le, de ezekkel elég kényelmetlen nagy mélységből vagy több helyről összeszedni a szükséges node-okat. A GetElementsByTagName segítségével név alapján megkapjuk a névre illeszkedő elemek listáját, tetszőleges szintről indulva és rekurzívan bejárva a dokumentumtöredéket.

A legkényelmesebb és legrugalmasabb megoldás a SelectSingleNode és a SelectNodes metódusok használata, amelyek a paraméterként átadott XPath kifejezés alapján válogatnak le egy vagy több node-ot. A következő példában megtekinthetjük az előbbi metódusok egy részét az alábbi dokumentum feldolgozására *(a.xml)*:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<NetAcademia>
   <soci labmeret="45">Soczó Zsolt</soci>
   <marci labmeret="49"></marci>
</NetAcademia>
```

Az üzleti probléma: Soczó Zsolt lába megdagadt a karácsonyi bejglitől, így 46-osra változott. Fóti Marcell lába összement a síbakancstól, így neki kettővel csökkent a lábmérete. Ráadásul a dokumentumból hiányzik az utóbbi neve, így pótoljuk azt. A kritikus üzleti folyamat implementációját DOM-mal az alábbi programrészlet szemlélteti *(a körítés ugyanaz, mint az előbbi példában)*:

```
// XmlDocument2.cs

XmlDocument d = new XmlDocument();
string docSource = "a.xml";
string docTarget = "b.xml";
d.Load(docSource);

//soci elem labmeret attributumának
//kikeresése
string xp = "/NetAcademia/soci/@labmeret";
XmlNode n = d.SelectSingleNode(xp);
//és módosítása
n.Value = "46";

//marci elem labmeret attributumának
//kikeresése property-kkel
//<NetAcademia>
```

```
XmlElement gyoker = d.DocumentElement;
//[0]:<soci>, [1]:<marci>
XmlNode e = gyoker.ChildNodes[1];
//@labmeret
XmlAttribute a = e.Attributes[0];

//XML dokumentumok kezelésekor nem string
//típusú adatok konverziójára ne a szoká-
//sos kasztolást ill. System.Convert osz-
//tályt használjuk, hanem az XmlConvert-
//et. Ez az XSD spec-nek megfelelően
//formáz illetve értelmez.
int klab = XmlConvert.ToInt32(a.Value) - 2;
a.Value = XmlConvert.ToString(klab);

//A marci elem tartalmának megkeresése
XmlNode m = d.SelectSingleNode("//marci");
XmlText t = d.CreateTextNode("Fóti Marcell");

//Az elem szöveges tartalama logikailag
//az elem gyermek node-ja.
m.AppendChild(t);

d.Save(docTarget);
```

A futtatás eredménye:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<NetAcademia>
  <soci labmeret="46">Soczó Zsolt</soci>
  <marci labmeret="47">Fóti Marcell</marci>
</NetAcademia>
```

A kimenet nem egy nagy ömlesztett xml dokumentum, hanem egy szépen megformázott és tabulált doksi *(nem én rendeztem be kézzel)*!
Ha sémával szemben ellenőrizve *(validating)* akarunk betölteni DOM-ba egy XML dokumentumot, akkor a Load() metódusnak nem egy fájlnevet, hanem egy inicializált XmlValidatingReader objektumot adunk át. Erről még fogunk értekezni egy későbbi számban, amiben az XML Sémával foglalkozunk.

XPathNavigator

A fenti DOM implementáció alternatívájaként a .NET biztosít egy másik megoldást is XML dokumentumok bejárására az XPathNavigator osztály formájában, ami a System.Xml.XPath névtérben található.
Ez általában nem olvassa be az egész dokumentumot a memóriába, csak azokat a részeket, amelyekre tényleg rányavigálunk. Persze ő is az XmlReader-re épít, csak nem olvassa be mohón az egész dokumentumot. Ezzel nyilvánvalóan jelentős időt és memóriát spórol meg, ha csak a dokumentum egy részével foglalkozunk. Az XSLT transzformációs osztály éppen ezért szívesen együttműködik vele, lehetővé téve nagyobb dokumentumok gazdaságos átalakítását. De erről kicsit később.
Az XPathNavigator is egy absztrakt alaposztály. A használatához valahonnan szereznünk kell egy konkrét megvalósítást, ami mögött valamilyen valódi adatforrás áll. Három olyan osztály van, ami képes egy inicializált XPathNavigator készítésére, az XmlDocument *(DOM, eddig erről volt szó)*, az XPathDocument és az XmlDataDocument. Az utolsóval a következő számban foglalkozunk, most nézzük meg a másodikat, az XPathDocument-et.
Az XPathDocument egy teljesítményoptimalizált, csak olvasható XML cache, amelyről lekérhetünk egy XPathNavigator példányt az xml forrás gyors bejárására. Tehát ő a DOM-mal ellentétben nem tölti be a teljes xml doksit, csak a hivatkozott részeket olvassa fel, de azokat eltárolja *(cache-eli)*, hátha később még el akarjuk érni. Könnyen el tudjuk képzelni, hogy például egy olyan XSLT transzformáció, ami csak a forrás egy kis részével foglalkozik, hatékonyan tud táplálkozni az XPathDocument-ből.
Nézzünk egy egyszerű példát a használatára!

```
1 XPathDocument doc =
2 new XPathDocument("a.xml");
3 XPathNavigator nav =
4 ((IXPathNavigable)doc).CreateNavigator();
5 XPathNodeIterator iterator =
6 nav.Select("/NetAcademia/*");
7 while (iterator.MoveNext())
8   Console.WriteLine(iterator.Current.Name);
```

Létrehozunk egy XPathDocument objektumot, ami majd az a.xml-t fogja feldolgozni *(2)*. Arról az IXPathNavigable interfészén keresztül tudunk lekérni egy XPathNavigator referenciát *(4)*, ami az a.xml-t járja majd be.
Az XPathNavigator Select() metódusában XPath kifejezés használatával kijelöljük a feldolgozni kívánt xml részeket *(6)*. A metódus kimenete XPathNodeIterator típusú referencia, amelyen keresztül MoveNext()-tel végiglépkedhetünk a visszakapott xml elemeken *(7)*. Ez tulajdonképpen egy node-halmaz. Az iterátor Current jellemzője újra csak egy XPathNavigator referenciát ad vissza, ami az aktuális node-on áll. A Name jellemző pedig ennek a node-nak a nevét adja vissza, de természetesen további információk is lekérhetők az aktuális node-ról.
Kimenet:

```
soci
marci
```

Látszik, az hogy XPathNavigator egy kicsit olyan, mintha a DOM-beli XmlNode-ként viselkedne, vagy mintha mindig egy xml node képe látszana rajta keresztül. A *(3)* sorban ez az egész dokumentumot reprezentáló node, az iteráció *(8)* során pedig az éppen aktuálisan elért node *(soci, marci)*.
Úgy is elképzelhetjük a navigátort, mint egy kurzort a forrás xml-en. Lehet mozgatni, és mindig lekérdezhetjük éppen melyik node-on áll. A Select() mellett bejárhatjuk a navigátort egyéb mozgató metódusokon keresztül is. Ezek részben hasonlítanak a DOM-os bejáró metódusokra. Például a navigátorban egy xml elemen állva MoveToFirstChild() elmozgatja a kurzort a legelső gyermekelemére. Ehhez hasonló a DOM FirstChild jellemzője, ami visszaad egy referenciát az aktuális node legelső gyermekelem node-jára. A MoveToNext() továbblép az xml forrásban = DOM nextSibling().
DOM kontra navigátor. Hogy melyiket használjuk rajtunk áll, ám dokumentumtöredékek feldolgozásánál mindenképpen érdemes fontolóra venni az XPathNavigator-t. Ez vonatkozik az előbbi „kézi" bejárásra és az XSLT transzformációkra is.

XSL transzformációk

Az egyik gyakran használt művelet xml források transzformációja valami más xml vagy egyéb szöveges kimenetté.
Ha adott a forrás xml fájl és az XSLT fájl, akkor transzformáció végrehajtása nagyon egyszerű:

```
//XslTransform1.cs részlet
using System.Xml.Xsl;
```

```
...
XslTransform s = new XslTransform();
xs.Load("traf1.xsl");
xs.Transform("atrafo.xml", "atrafoki.xml");
```

A példafájlokat illetve a teljes forrást a **[2]** címen lehet letölteni. Az XslTransform osztály azért ennél jóval okosabb. Gyakori feladat, hogy paramétereket kell átadni az XSL transzformációnak, így az a paraméterektől függő feldolgozást hajt végre a forrásdokumentumon.

```
1  // XslTransform2.cs részlet
2  //XML forrás
3  XPathDocument doc =
4  new XPathDocument(
5  GetXmlReaderCEU("atrafo.xml"));
6
7  XslTransform xs = new XslTransform();
8
9  //XSLT forrás
10 xs.Load(GetXmlReaderCEU("traf2.xsl"));
11
12 //Ebben lesznek a paraméterek
13 XsltArgumentList xslArg =
14 new XsltArgumentList();
15 //Két paramétert hozzáadunk
16 xslArg.AddParam("kislab", "", 46);
17 xslArg.AddParam("nagylab", "", 50);
18
19 //Ebbe megy a trafo kimenete
20 XmlTextWriter writer =
21 new XmlTextWriter("atrafoki2.xml",
22 Encoding.GetEncoding("ISO-8859-2"));
23
24 xs.Transform(doc, xslArg, writer);
```

(3)-(5) sorokban a forrás transzformálandó dokumentumot rendeljük hozzá egy XPathDocument példányhoz. Nem XmlDocumenthez, már tudjuk miért. A GetXmlReaderCEU() egy kis kényelmi metódus, ami XmlTextReader-t hoz létre a paraméterként megadott fájlra, ISO-8859-2, az Center European kódolást feltételezve. Ez szükséges, ha a forrás doksik vagy az XSLT-k ékezetes karaktereket tartalmaznak, és nem Unicode karakterenként vannak tárolva. Így néz ki:

```
public static XmlTextReader
GetXmlReaderCEU(string path)
{
   StreamReader reader =
   new StreamReader (path,
   Encoding.GetEncoding("ISO-8859-2"));
   return new XmlTextReader(reader);
}
```

(7)-ben létrejön a transzformációs objektumunk, *(10)*-ben betöltjük az XSLT-t. A transzformációban felhasznált paramétereket egy XsltArgumentList objektumban rakjuk össze és adjuk át transzformációnak *(13-14)*. Az AddParam() hívás a paraméter nevét, névterét és értékét várja bementül *(16-17)*. Az átalakítás eredménye egy XmlTextWriter-re megy *(20-22)*, ahol a karakterkódolásra ismét ügyelni kell. Végül a *(24)*-es sorban végrehajtjuk a mágiát, a forrás, a paraméterek és a cél felhasználásával. Az XSLT eleje így fest *(traf2.xsl)*:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 version="1.0">
  <!-- Ezeket a paramétereket töltjük
       ki híváskor. -->
  <xsl:param name="kislab"/>
  <xsl:param name="nagylab"/>
...
```

Később a transzformációkban felhasználjuk a két paraméternek átadott értéket. A teljes XSLT letölthető a **[2]** címről.

Ínyencek *(perverzek)* oldala

Most pedig jöjjön egy kis ravaszság. Hogyan írjunk kiegészítő függvényt XSLT-ben, valamilyen menedzselt nyelven? Aki ismeri az MSXML script elemét, annak ez nem okoz nagy meglepetést, de aki még nem látott ilyet, az ámuljon. Már megint az XPathNodeIterator! A forrás node-set-jét így kapjuk meg paraméterül. Futtatás például az XslTaransform1.cs átírásával *(trafo3.xsl)*. A kimenet: 49.

```
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:msxsl="urn:schemas-microsoft-com:xslt"
 xmlns:naca="http://www.netacademia.net"
 version="1.0">
  <xsl:output method="text"
   encoding="ISO-8859-2"/>

  <msxsl:script language="C#"
   implements-prefix="naca">
    <![CDATA[
    int LegnagyobbLab(
          XPathNodeIterator iterator)
    {
      int maxprice = 0;
      while (iterator.MoveNext())
      {
          int price = System.Convert.ToInt32(
          iterator.Current.Value);
          if ( maxprice < price)
              maxprice = price;
      }
      return maxprice;
    }
    ]]>
  </msxsl:script>

  <xsl:template match="/">
  <xsl:value-of
select="naca:LegnagyobbLab(//@labmeret)"/>
  </xsl:template>
</xsl:stylesheet>
```

Soczó Zsolt
Zsolt.Soczo@netacademia.net

A cikkben szereplő URL-ek:
[1]: .NET Reflector
http://www.aisto.com/roeder/dotnet
[2]: Letölthető példák
http://technet.netacademia.net/download/xml

SQL Server
Join algoritmusok

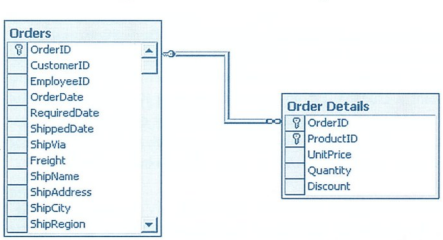
Az elmúlt alkalommal megvizsgáltuk a Query Optimizer működésének főbb alapelveit és kiveséztünk néhány egyszerű végrehajtási tervet *(query plan)*. Most – ígéretemhez híven – a kapcsolási, avagy join stratégiák kerülnek sorra. Joggal kérdezhetik, hogy minek ez nekünk, hisz mindig minden automatikusan dől el az SQL Serverben. Ez igaz. De hogy **milyen döntés** születik automatikusan, azt súlyosan befolyásolhatják a mi korábbi tervezési döntéseink.

S nem mellesleg: a Designing vizsgán a kérdések jó 15%-a a végrehajtási tervekkel foglalkozik!

Egy-egy adatbázis még a legegyszerűbb esetben is több táblából áll, hiszen az adatbázistervezés egyik fő célja, a redundancia csökkentése a legkönnyebben úgy valósítható meg, hogy az ismétlődő adatokat a főtáblából kiemeljük, és külön táblába tesszük. Itt és most nem szeretnék elmerülni az adatbázistervezés rejtelmeiben *(erre is sort kerítünk a későbbiekben)*, hanem kész, megtervezett és feltöltött táblák segítségével mutatom meg, mit tesz az SQL Server, amikor egy lekérdezés egynél több táblát érint. *(Az egyszerűség kedvéért az „egynél több" ebben a cikkben mindig kettő. Nem fogunk hat-nyolc táblát összedzsoinolni, mert azzal feleslegesen bonyolítanánk a gondolatmenetet. Maga az SQL Server is így tesz: ha begépelünk neki egy nyolctáblás dzsoint, akkor a feladatot lebontja páronkénti összekapcsolásokra.)*

Használjuk az egyszerűség kedvéért az SQL Serverben *(is)* megtalálható Northwind adatbázist, s annak is két legnagyobb tábláját, az **[Orders]** *(rendelések, 830 sor)* és az **[Order Details]** *(rendelések tételei, 2155 sor)* táblát. Éles adatbázisban a kétezer sor természetesen nem mennyiség, de kísérleteinkhez ez is elegendő lesz. Amint az az alábbi ábrán látható, az **[Orders]** tábla elsődleges kulcsa az **[OrderID]** mező, melyen egy *(clustered)* index csücsül. Az **[Order Details]** tábla elsődleges kulcsa összetett, két mezőt foglal magában: az **[OrderID]** és a **[ProductID]** mezőket. Az ehhez tartozó „gyári" index itt is clustered.

(Az igazsághoz hozzátartozik, hogy az **[Order Details]** *tábla kegyetlenül, feleslegesen és bután agyon van indexelve, mert az* **[OrderID]** *mezőn összesen három index van, s ezek közül kettő tökéletesen egyforma nonclustered index. Ez utóbbi kettő bátran törölhető lenne. Egyáltalán minek hozták létre?)*



A Northwind adatbázis két legnagyobb tábláját használjuk joinkísérleteink végrehajtására

A két tábla úgy kapcsolódik egymáshoz, hogy az **[Order Details]** táblán van egy idegen kulcs, mely a két táblát az **[OrderID]** mentén köti össze. Ezt a relációt szokták one-to-many *(egy a sokhoz)*, vagy apa-fiú kapcsolatnak is hívni.

De itt és most azonnal számoljunk le egy gyakori tévhittel: az ábrán a táblák közötti **referenciális integritási** szabályt látjuk, mely csak azt határozza meg, hogy milyen rekordok **születhetnek** a két táblában, illetve milyen **módosítás** áldozatául eshetnek a meglévő adatok. Magyarán *(és iszonyú röviden)*: gyermekrekord nem születhet apuci nélkül, és aput nem lehet legyilkolni mindaddig, amíg élnek a gyermekei. Azonban a referenciális integritási szabályok semmilyen korlátozást nem jelentenek a lekérdezésekre vonatkozólag. A fennálló integritási szabály nem akadályozhat meg minket abban, hogy olyan lekérdezést írjunk, ami a két táblát például az **[Orders].[ShipVia]** és az **[Order Details].[Quantity]** mentén kapcsolja össze – legyen ez bármilyen értelmetlen feladat is *(lásd később)*. Pusztán az értelmes felhasználás miatt szokás gyakorta ugyanúgy joinolni egy lekérdezésben, mint ahogy a referenciális integritási szabályok is állnak.

Essünk neki

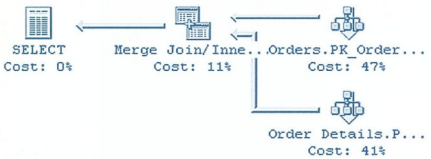
Tákoljunk össze egy olyan lekérdezést, mely mindkét táblát megmozgatja, és vizsgáljuk meg ennek végrehajtási tervét. Kérjük le az összes kapcsolódó rekordot!

```
SELECT * FROM
  [Orders] INNER JOIN [Order Details] ON
    [Orders].[OrderID] = [Order Details].[OrderID]
```

Gondolkodjunk az SQL Server fejével! Hogyan lehetne előállítani a fenti lekérdezés eredményét, mely minden egyes **[Order Detail]** sorhoz keresi annak „apukáját" az **[Orders]** táblában? Két stratégia is kínálkozik. Az egyik, hogy végigszaladunk az **[Order Details]** táblán, és soronként kikeressük hozzá a megfelelő aparekordot. A másik módszer szerint az aparekordokon gyalogolunk végig egyesével, és megkeressük minden sorhoz az összes gyermeket. Programozói szakzsargonnal: két egymásba ágyazott ciklust használunk; az egyik táblán végigszaladunk egy ciklussal *(ez a külső, outer ciklus)*, s ennek minden állomásán lefuttatunk egy másik ciklust *(belső, inner)* a másik táblán. Ezt a stratégiát Nested Loopnak hívják, és a Query Planen így fest:

Nested Loop

Az SQL Server úgy állapítja meg, hogy melyik ciklus melyik táblán fusson, hogy rá se pillant a referenciális integritásra. Az esetek elsöprő többségében a kisebb *(kevesebb rekordszámú)* táblán fut a külső ciklus – így „olcsóbb". Tehát a mi esetünkben az **[Orders]** lesz a külső, és az **[Order Details]** - mely vagy háromszor annyi sort tartalmaz – lesz a belső. Lássuk a tervet!



A fenti joinos SELECT futtatási terve. Nested Loopra számítottunk, ehelyett Merge Joint kaptunk

Hümm. Ez bizony nem Nested Loop! Ez egy teljesen másik join stratégia, az úgynevezett Merge Join. Az igazat megvallva én ezt előre tudtam. Az egymásba ágyazott ciklusokat használó Nested Loop ugyanis a legalapvetőbb, és legegyszerűbb stratégia. Emlékeim szerint SQL Server 6.5-ig bezárólag csak ez a stratégia létezett. A következő verzió, a 7.0 – mely kenterbe verte elődjét az összes teljesítményteszten – többek között azért tudott sokszorosan nagyobb teljesítmény nyújtani óriási adatbázisok esetén, mert két újabb join stratégiát okoskodtak ki a redmondi fejlesztők: a Merge és a Hash joint. Ezek megértéséhez a megfelelő alapok már rendelkezésre állnak, hisz tavaly októberben kimerítő cikket írtam a hash algoritmusokról *(a húsdarálós)*.

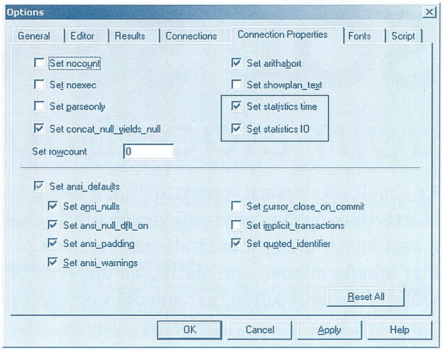
Mi a baj a Nested Looppal?

Két nagy baj van vele: az, hogy Nested és az, hogy Loop. Az egymásba ágyazott ciklusok miatt ugyanis a belső ciklus annyiszor fut le, ahány lépése van a külsőnek. Azaz, ha a külső ciklus 830 sort talál, akkor a belső táblát bizony 830-szor „vesszük kézbe". Ha a táblák még nagyobbak, a helyzet egyre csak romlik. Tegyünk egy kísérletet, mérjük össze a Nested Loop és a Merge Join sebességét és viselkedését ezen a két táblán. Ehhez elsőként állítsuk be a Query Analyzert úgy, hogy mérje a futtatási időket, valamint hogy hányszor kell a táblákhoz nyúlni *(Tools->Options->Connection Properties fül)*: pipáljuk be a **Set statistics time és a Set statistics IO** opciókat!

Vizsgára készülők figyelmébe!
Minden, ami egérrel bekattintható, az Transact SQL parancsok segítségével is beállítható. Ezeket érdemes vizsgára menetel előtt áttekinteni, bemagolni, majd a vizsga után lazán elfelejteni. Jelen esetben a

```
SET STATISTICS TIME ON
SET STATISTICS IO ON
```

Parancsokkal érhető el ugyanaz a hatás.



Felkészítjük a Query Analyzert kétféle teljesítménystatisztika gyűjtésére. Az IO mérés alapvetően a 8k-s lapok felhasználásának számlálására való

Ha ezekkel a beállításokkal lefuttatjuk a cikk eddigi egyetlen, fentebb olvasható joinos SELECT lekérdezését, a következő eredményeket kapjuk:

```
(2155 row(s) affected)

Table 'Order Details'. Scan count 1, logical reads 10, physica
Table 'Orders'. Scan count 1, logical reads 21, physical reads

SQL Server Execution Times:
   CPU time = 90 ms,  elapsed time = 443 ms.
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.
```

Grids | Messages

Táblázatos kimenet esetén a teljesítménymérés eredménye a Messages fülön jelenik meg. Ez itt a Merge join eredménye

Feljebb olvasható az IO statisztika, alant a futási idők.

- **Scan count:** ennyiszer kellett hozzányúlni az adott táblához. Könnyedén leolvasható, hogy mindkét táblát egyszer „fogtuk meg"
- **Logical reads:** az olvasás során feldolgozott 8k-s lapok száma. 10 és 21. Nem rossz!
- **Physical reads:** hány olyan 8k-s lap volt, amit a lemezről kellett felemelni – merthogy nem volt a kessben. Miután én már ikszedik alkalommal futtattam ezt a lekérdezést, mindkét tábla bekúszott a kessbe, így ez a mutató mindkét táblánál nulla.(Az ábrán nem látszik kilóg).

A végrehajtási idők közül a CPU time számít: ez 90 ms. Ezekkel az eredményekkel kellene összevetni a Nested Loop stratégia eredményeit, de ehhez rá kellene venni az SQL Servert, hogy hagyja a Merge joint a csudába. Ezt úgynevezett optimizer hintekkel *(súgásokkal)* tehetjük meg. De vigyázat! Optimizer hintet csak és kizárólag tesztelési célból használjunk, mert ha belevarrjuk az alkalmazásba, óriási hibát követünk el! Lásd az előző cikkem elrettentő példáit!

Erőszakoskodás Optimizer hintekkel

A Books Online részletesen taglalja az optimizer hinteket, melyik mire jó, hogyan lehet segítségükkel zárolást, indexhasználatot, dzsoinstratégiát rálőcsölni szegény SQL Serverre. Ha a fenti egyetlen SELECT végére odabiggyesztjük, hogy

```
SELECT * FROM
  [Orders] INNER JOIN [Order Details] ON
    [Orders].[OrderID] = [Order Details].[OrderID]
OPTION (LOOP JOIN)
```

akkor ugyanez a lekérdezés Nested Loop stratégiával fog lefutni. Az alábbi táblázat segítségével könnyedén összehasonlítható a két joinstratégia teljesítménykülönbsége. A könnyebb összehasonlítás kedvéért csak az egyik tábla, az [Order Details] adatait foglaltam össze:

	Merge join	Nested Loop
Scan count	1	830
Logical reads	10	1672
CPU time	90	130

Világosan látszik a Nested Loop katasztrofális teljesítménye, melynek oka: egymásba ágyazott ciklusok nagy táblákon. A futásidő csak azért nem változott még ennél is drámaibban, mert a két tábla viszonylag pici, és ráadásul kessben van. De nem eresztenem rá a Nested Loopot több millió soros táblákra!
Félreértés ne essék: ez nem azt jelenti, hogy a Nested Loopot tűzzel-vassal irtani kell, mert megvan a maga szerepe. Ha a két tábla közül akár az egyik icipici, máris sokkal gyorsabb, mint akár a Merge, akár a Hash!
Ha a fenti WHERE feltétel nélküli SELECT-et ellátjuk szűrőfeltétellel:

```
SELECT * FROM
  [Orders] INNER JOIN [Order Details] ON
    [Orders].OrderID = [Order Details].OrderID
    WHERE
[Orders].OrderID=10248
```

akkor az **[Orders]** tábla máris pici lesz *(3 sort talál a lekérdezés)*, és az SQL Query Optimizer már hajítja is el a Merge joint, és automatice visszavált Nested Loopra.
Ha most ezt a WHERE feltételes lekérdezést erőszakoljuk meg, és szándékosan Merge joinba kényszerítjük, szintén katasztrófa lesz a vége. Nem is akármilyen katasztrófa! Kapunk egy 8622-es hibaüzenetet, mely szerint:

„Query processor could not produce a query plan because of the hints defined in this query. Resubmit the query without specifying any hints and without using SET FORCEPLAN."

Ajjaj!

Vigyázz! Warning! Pozor!
Soha ne varrj be Optimizer hinteket a kódodba! Isten tudja, mi lesz belőle később!

Mire nem jó a Nested Loop?
a Nested Loop stratégia használhatatlan many-to-many lekérdezéseknél (cross joinnál, vagy másnéven cartesian productnál)

Merge join

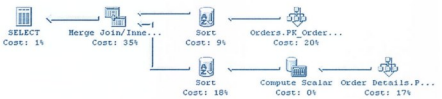
Ha már annyi szó esett a Merge Join fantasztikus teljesítményéről, itt az ideje, hogy megtudjuk, mit csinál másképpen, mint a Nested Loop. Azt már a fenti táblázatokból is láthattuk, hogy mindkét táblán csak egyetlenegyszer megy végig. Vajon hogy csinálja, hogy nem kell visszafordulnia? Ez csak egy módon képzelhető el: ha a rekordok mindkét táblában rendezetten érhetők el, vagyis vagy

- van jó index, mely segít a joinmezőkön végigfutni, vagy
- a Merge előtt lefut egy logikai sorbarendezési művelet *(Sort)*

A Merge joinnak két lába van. Az egyikkel az egyik tábla rekordjain csoszog előre, a másikkal a másik táblán. Előrelép az egyikkel, talpa alatt egy kulcsmező, melynek értéke mondjuk 10248. Ezután a másik lábát addig csúsztatja előre a másik táblán, amíg ott is 10248 van a talpa alatt. Ha túlfut, megáll, és megint az első lábát kezdi húzni mindaddig, amíg az első táblában olyan rekordokat talál, mint amelyiken a másikon áll. Mint ahogyan az ember a jégen megy. Egyik lábát addig tolja előre, ameddig csak bírja *(vagy be nem szakad)*, s utánahúzza a másikat.
Mivel jó index mentén csoszogó Merge joint már láttunk az előző oldalon, most nézzük meg, hogyan fest egy indexmentes, vagy használhatatlan indexekkel „segített" lekérdezés futtatása. Egy extrém példával együtt említettem, hogy a referenciális integritásnak vajmi kevés köze van a join stratégiákhoz, azt joinulunk össze és azzal, amit csak akarunk. Akár teljesen értelmetlen dolgokat is összekapcsolhatunk:

```
SELECT * FROM [Orders]
  INNER JOIN [Order Details] ON
    [Orders].[ShipVia] = [Order Details].[Quantity]
```

Ez a lekérdezés kilistázza azokat a rekordokat, ahol a rendelések szállítási módja egyenlő a szállítási mennyiséggel :-)
A végrehajtási tev elgondolkodtató:



Ha nincs (jó) index, a Merge Join csinál magának. Ez a Sort műveletből látható. Tekinthetjük dinamikus indexnek, melyet a lekérdezés végén elhajítunk.

Mintha indexet használna a sorbarendezés *(Sort)* művelet előtt! Ez azonban optikai csalódás. Az ábra jobb szélén ugyanis Clustered Index Seeket látunk, ami a közönséges táblavégigolvasás *(Table Scan)* **helyett** jelenik meg a táblán, melynek fizikai sorrendjét a clustered index határozza meg.

Mire nem jó a Merge join?
A Merge Join használhatatlan, ha a join operátor nem egyenlőségjel *(hanem <, >= stb.)*

Hash join

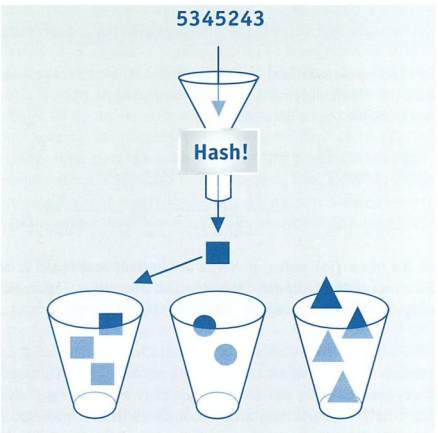
És végül, de nem utolsósorban, itt a hash join. Amilyen nehéz volt elképzelni, hogy mi kellhet a Nested Loopon kívül, épp olyan nehéz felfogni, hogy mi maradt, amit a Merge joinnal sem lehet megoldani. Pedig van ilyen join probléma: hatalmas táblák index nélkül! Ilyen nincs ugye? De bizony van. Még véletlenül, hanyagságból is előfordulhat. Vagy egy olyan ad-hoc lekérdezést kér a marketingfőnök, ami a vevők lábméretét összedzsoinolja a raktárkészlet újrarendelési küszöbével :-) Egyszóval olyan lekérdezés, amire nincs index, és nem is érdemes csinálni, mert több órán keresztül tart, és utána nem kell semmire.
Mi a hash? Egy húsdaráló. Mit darál? Bármit. Mi lesz belőle? Fasírt. Digest. Egy olyan érték, mely jellemző arra a bemenő adatra, amit bedobtak a darálóba.

Mi lehet a célunk két irdatlan nagy, indexmentes tábla összejoinolásakor? Hogy ne kelljen unos-untalan végigtekerni az egyik táblán, hogy párt találjunk hozzá a másikban. Erre találták ki az indexeket, dehát nincs nekünk olyan.

Elő kellene venni a tech.net tavaly októberi számát, ugyanis ott, a Hash! című cikkben részletesen elmeséltem a nagy halmazokban történő keresés gyorsítását Pistikével, és a tanító nénivel, akinek Pistike rágógumit rakott a székére. Ha az osztályba tízmillióan járnak, sokkal gazdaságosabb keresést biztosít, ha Pistike önként jelentkezik, mint ha mind a tízmillió kis csibészt végigkérdezem. Mivel Pistikének esze ágában sincs jelentkezni, a tízmillió csirkefogó gyereket bizonyos jellemzőjük alapján csoportokba *(hash buckets)* rendezzük, és csak a „gyanús" csoportokat vizsgáljuk át. A többiek hazamehetnek.

Ez történik a hash joinnál

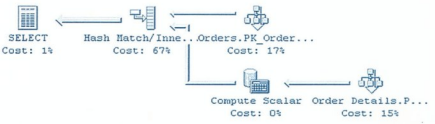
Van két táblánk, külön-külön foglalkozunk velük. Először a kisebbiken megyünk végig, s a joinban használt *(kulcs?)*mezőre ráeresztünk egy hash algoritmust *(hogy melyiket, azt sajnos nem találtam meg az általam átböngészett szakirodalomban)*, majd az így előállt kulcs-hasheket vödrökbe *(bucket)* szortírozzuk.



Hash buckets, melyekbe az egyedi hash értékek szétválogatódnak. Így egy rekord keresése csak egy vödör átbogarászását igényli, nincs szükség a teljes tartomány átfésülésére

Amikor ez megvolt, jöhet a második tábla feldolgozása. Szépen, egyesével kiszámítjuk minden rekord kapcsolómezőjéhez a hash értéket, s megnézzük, vajon melyik vödörben lehet a párja. Az adott vödör tartalmát azután egyesével átfésüljük, míg meg nem találjuk a kívánatos érték*(ek)*et. Ilyen egyszerű!

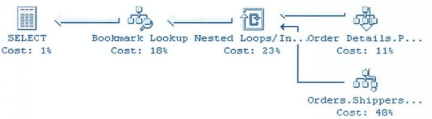
Mit is nyertünk ezzel? Azt, hogy bár két hatalmas, indexmentes táblánk volt, mégis elegendő volt egyetlenegyszer végigmenni mindegyiken, hogy megtaláljuk a joinnak megfelelő sorpárokat. Íme itt egy hash végrehajtási terv, melyet a megfelelő optimizer hint segítségével kényszerítettem ki:



Hash join a két táblán. Csalással (hint) készült, mert a táblák nem elég nagyok ahhoz, hogy ezt válassza az SQL Server

Mire nem jó a Hash join?

Tavaly októberben még nem tudtam, hogy az SQL Serverben felhasznált hash algoritmus egyedi értékeket ad-e. Most már tudom. Egyedi értékeket ad. Emiatt sajnos a hash joinra ugyanazok a korlátozások érvényesek, mint a merge joinra: a joinfeltételnek egyenlőségnek kell lennie. Ha nem az, akkor gigantikus táblák esetén is marad a jó öreg Nested Loop, bár annak egy érdekes változata, amikor a ciklusok csak a kulcsokat pároztatják össze, mert az összes rekord nem fér be a memóriába. Ha kész a kulcsokra a ciklus, utólag, külön lépésben halássza-vadássza össze a teljes rekordokat. Álljon itt mementóul egy Bookmark Lookupos, Nested Loop, hogy érzékeljük a Query Optimizer határtalan bölcsességét:

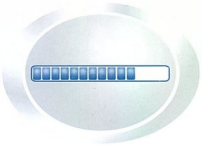


Nested Loop, de a teljes eredményhalmaz nem fért be a memóriába. Ilyenkor először csak a kulcsokat loopolja össze, majd Bookmark Lookuppal szedegeti fel a rekordokat

Mostanra elég sokmindent végigvettünk a Query Optimizer lehetőségei közül. Nem célom az összes kis ikon végigmagyarázása, de ha találok még érdekességet a témában, ígérem, megírom. Legközelebb azonban a zárolások következnek. Mi az a Deadlock?

Fóti Marcell
MCDBA
marcellf@netacademia.net

MSDownload – Patchwork



2001 október közepétől a Netacademia Kft üzemelteti Magyarország egyik legnagyobb letöltőközpontját. A Microsoft Magyarország megbízásából elkészült oldalak az eddig megszokott tartalommal, de eltérő formában jelentkeznek. Az új megjelenés könnyű kereshetőséget, világos, áttekinthető formát biztosít a látogatók számára. Az innen letölthető programok közül csemegézünk ebben az új rovatban.

A Microsoft hivatalos letöltőközpontjában **[1]** időről-időre megjelennek olyan anyagok, amelyek valamilyen módon kiemelkednek a szokásos ingyenanyagok közül. Ezekre szeretnénk felhívni a figyelmet ebben a rovatban.

Halmozott javítás Internet Explorerhez *(MS01-55)*

2001. november 8-án a Microsoft egy olyan javítást adott ki, amely nem javított ki minden biztonsági rést az IE 5.5 és IE 6.0 böngészőprogramokban. Rövid időn belül be is vonták az így kiadott biztonsági javítást, majd november 13-án már egy újabb verzióval találkozhattunk, mely már valóban orvosolja az alábbi hibákat.

Ezek a rések egyrészt abból adódnak, ahogy az Explorer a cookiekat kezeli. A „sütiken" keresztül felépíthető volt egy olyan ellenőrizhetetlen kapcsolat, amellyel mélyebben fekvő információkhoz hozzá lehetett férni, sőt azokat bizonyos esetben módosítani is lehetett. Másrészt néhány weboldal érzékeny információkat tárol egy-egy cookie-ban *(például felhasználónév-jelszó páros)*, lehetővé téve ezen személyes információkkal való visszaélést.

A fejlesztők ebben a hotfixben kijavítanak egy másik hibát is, amely az MS01-51 számú biztonsági javításban orvosoltak egy újabb fajtája, nevezetesen a „fertőzések pontnélküli URL-eken keresztül" fedőnevű.

Ha egy weboldalt a pontnélküli IP címével hívunk meg *(például http://031713501415 a http://207.46.131.13 helyett)*, és a megadott cím létezik, akkor az Internet Explorer nem ismeri fel, hogy a megadott cím egy Internetes oldalt takar. Ehelyett úgy kezeli, mintha intranetes lenne és ezért helytelenül az intranet zónában nyitja meg. Így pedig az adott oldalnak kevesebb védelmi előírásnak kell megfelelnie. Ez a hiba az Explorer 6.0-át már nem érinti.

Azok, akik a november 8-án kiadottaknak megfelelően jártak el, és Explorer-ükben letiltották az Active Scripting funkciót, újra engedélyezhetik azt ezen javítás telepítése után.

Újabb rések betömése az Internet Explorerben *(MS01-58)*

Újabb biztonsági réseket javíthatunk ki Explorerünkön, ha a Microsoft által 2001. december 13-án kiadott hotfixet telepítjük. Ez tartalmazza az eddig ismert minden hiba javítását, amit az Internet Explorer 5.5 SP2-ben és az Internet Explorer 6.0-ban felfedeztek. Ezentúl három új támadási felületet tudunk vele megszüntetni.

1. Az egyik illetéktelen behatolásra éppen a HTML fájlok felépítése adja meg a lehetőséget. A HTML fájlok fejrészében több olyan adatot is elhelyezhetünk, ami az adott oldalról közöl különböző információkat. Ezek közül kettő, a Content-Disposition és a Content-Type mezők azt a környezetet hivatottak leírni, hogy az oldal letöltése után az böngésző hogyan kezelje az átküldött információt. Ez azért veszélyes, mert így egy futtatható állományról az Explorer elhiszi, hogy ártalmatlan HTML fájllal áll szemben, és bármilyen figyelmeztetés vagy felhasználói jóváhagyás nélkül megnyitja azt. Így egy rosszindulatú HTML levél vagy weboldal megnyitásakor egy exe állomány kezd futni gépünkön, kisebb vagy nagyobb bosszúságot okozva. Ez a biztonsági hiányosság csak az IE 6.0-t használókat érinti, aki IE 5.5 SP2-vel rendelkezik, védve van az efféle támadásoktól.

Mivel a legtöbb ilyen behatolási próbálkozás az Internet vagy intranet zónából érkezik, ezért ebben a két zónában az alapértelmezett File Download *(fájl letöltés)* engedélyezést tiltásra kell átállítani. Így kivédhetünk minden hasonló esetet.

2. A másik biztonsági rés a már kiadott MS01-15-ös számú biztonsági javításban közöltek egy újabb verziója. Ez egy rosszindulatú webszerkesztőnek megengedi, hogy két böngészőablakot nyisson. Az egyiket a weboldal domain-jében, a másikat pedig a felhasználó rendszerében. Így minden olyan fájlhoz hozzáfér, amit egy böngészőben meg lehet nyitni. Ha gépünket egy ilyen hívatlan látogató eléri, akkor bármilyen képet, szöveges állományt, vagy HTML fájlt képes megnyitni és elolvasni, amit eltároltunk. A többi fájltípushoz, mint például a bináris, futtatható állományok vagy Word dokumentumok, nem fér hozzá. Azon túl, hogy bizonyos fájlok teljes tartalmához hozzáfér, a fájlok pontos nevét és helyét is meg tudja állapítani. „Csak" olvasási joggal rendelkezik, tehát se törölni, se állományt létrehozni, módosítani vagy futtani nem tud. Ez a rés viszont már mind az IE 5.5-öt, mind az IE 6.0-t érinti.

3. A harmadik biztonsági hiányosságot a File Download *(Fájl letöltése)* ablakban fedezték fel. Amikor egy fájlt szeretnénk letölteni, akkor egy dialógusablak jelenik meg, ahol a fájl nevét a rendszer automatikusan kitölti. Bizonyos esetekben előfordulhat, hogy rendszerünket úgy próbálják megtámadni – legyen az weboldal vagy HTML formátumú levél –, hogy a fájlnév kicsit elferdítve kerüljön ebbe a mezőbe. Például megváltozik a kiterjesztés vagy akár csak egy betűnyi különbséget találunk. Ám ez elég, hogy egy korántsem biztonságos fájlt letöltsünk.

Az, hogy egy fájl letöltését engedélyezzük, soha nem azon alapszik, hogy milyen kiterjesztéssel rendelkezik az adott állomány, hanem azon, hogy a forrásban mennyire bízunk meg. Ezért megbízhatatlan forrásból ne töltsünk le fájlokat, még akkor sem ha a típusa egyébként megbízhatónak tűnik.

Borsi Katalin
bobo@netacademia.net

A cikkben szereplő URL-ek:
[1] http://msdownload.netacademia.net

Rendszerváltás
IPv6

Csak semmi aggodalom! Most nem hullanak fejek, nem cserélik le az utcatáblákat, és azt sem írják elő, hogy hány ágú, milyen színű csillagokkal díszíthetjük a karácsonyfát. Szerencsére másról, sokkal érdekesebbről és hasznosabbról lesz szó: az Internetről. Helyesebben annak, gyakorlatilag egyeduralkodó protokolljáról, az Internet Protokollról *(IP)*, amelynek jelenleg a 4-es számú verzióját használjuk.

A 4 bájtos címmezővel ábrázolható, mintegy 4 milliárd IP cím 20 évvel ezelőtt olyan nagynak tűnt a protokoll kidolgozói számára, hogy végül mégiscsak kevés lett. A bőség zavarának köszönhetően fordulhat elő ma olyan eset, hogy egyetlen egyetem nagyobb címtartománnyal rendelkezik, mint egy egész ország. Az osztályos IP világban nem lehetett szűkmarkúnak lenni: a C osztály sok szervezet számára kevésnek bizonyult, és már a B-vel is egyszerre 65536 cím veszett el – akkor is, ha csak 300 kellett volna. Az Internet robbanásszerű növekedése hamar jelezte, hogy nem lesz ez így jó, így vérmentes forradalom keretében felszámolták az osztályokat, az így létrejött, ma is működő világot pedig CIDR-nek *(Classless InterDomain Routing)* nevezték el. Ezenkívül a NAT *(Network Address Translation)* is segédkezet nyújtott, hiszen ekkor egyetlen cím mögé egy újabb Internet bújhat.

Az Internet rengeteg alkalmazást rejt magában. Hűtőgép, amely kaját rendel az áruházból; autó, amely kinyitja a garázsajtót, „csomagkapcsolt" TV; kommunikátor stb. Némelyiket már ma használjuk, és feltehetően a többi megszokása is csak idő kérdése. Bármennyire is úgy tűnik ma, hogy sosem lesz rájuk szükség, a következő generáció tisztelettel adózik majd előttünk, hogy ezek nélkül is képesek voltunk a túlélésre. Igény tehát van.

Amilyen gyakori ma a cég-, termék- és technológianevekben az e és az x betű, olyan gyakori lesz néhány éven belül a 6-os szám: ftp6, ping6, tracert6 stb. Ezek mind az IP új, 6-os számmal jelölt verziójára épülnek. Az IPv6 egyik legfontosabb tulajdonsága, hogy a címeket 16 bájtosra növelték, így Tannenbaum érzékletes példája szerint a Föld minden négyzetméterére 1000-nél is több cím jut. Ha az Egyesült Nemzetek előrejelzése szerint 20 év múlva 10 milliárd ember taposса ezt a bolygót, az akkor is fejenként több mint 1 millió cím.

Nyilvánvaló – hiszen nem lenne most miről írnom, ha nem így lenne, – hogy a Microsoft sem figyelte tétlenül az IPv6 fejlődését. A kutatási részleg, Draves és Zill személyében már 1996-ban bekapcsolódott az IETF szabványosítási munkájába. Először az NT 4.0 TCP/IP forráskódját írták át, de mivel ez nem minden területen adott kielégítő eredményt, egy módosított változattal álltak elő. Az MSRIPv6 1.0 1998-ban jelent meg. Draves és Zill kérésére a Microsoft Windows részlege nemcsak a kód megváltoztatását engedélyezte, hanem merőben szokatlan módon a forráskód nyilvánosságra hozását is. Elkezdődött egy új közösség kialakulása, szoftverrel, levelezőlistával, eszmecserével. 2000 elején letölthető volt a Windows 2000-hez a technology preview, vagyis amolyan bemutató jellegű termék. Ehhez már API is tartozott, tehát a vállalkozó kedvűek alkalmazásokat készíthettek hozzá. Végül, 2001-ben az XP bétában ott díszeleg a fejlesztői változat. Terveik között ott szerepel a .NET Server-hez kapcsolt IPv6 protokollverem kereskedelmi változata, amelyhez már a támogatás is dukál.

Ez a bámulatos lendületesség nem véletlen. Nagyon nagy a tét. A korábban felsorolt alkalmazási lehetőségek közül a mobil világ és az Internet konvergenciájából (3G: 3. generáció) születő kommunikátorszerű kütyük közeljövőbeli áradata a legvalószínűbb. Ebben a felvonásban még nem tisztázódtak teljesen a szerepkiosztások, de a jelek szerint teljesen világos, hogy a Microsoft nem statiszta akar lenni.

Némi általános vita jellemzi azt a kérdést, hogy az átállást a hardverrel vagy a szoftverrel kell-e kezdeni? A Microsoft mint szoftvercég, eléggé el nem ítélhető módon reagált a kihívásra. Huitema kidolgozta a 6to4 módszert, amivel minden egyes IPv4 címhez egyértelműen rendelhető IPv6-os prefix *(a hálózati maszk jogutódja)*. Eszerint minden IPv6-os implementációnak ismernie kell majd a 6to4 módszert: amikor ilyen célcímmel találkoznak, akkor az IPv6-os címből IPv4-est kell előállítani, és az eredeti IPv6-os csomagot az előállított IPv4-es célcímmel IPv4-es módon kell becsomagolni *(tunnelling)*.

Térjünk vissza egy kicsit a 3G-hez. A mobil világ alapvetően idegen az Internettől. Mert milyen dolog az, hogy valaki fogja magát, aztán egyetlen lépéssel egy másik országban lévő útvonalválasztó mögé kerül? Pedig ilyen bizony gyakran előfordul majd, hiszen nemhogy tiltani nem akarják a szolgáltatók az efféle skandallumot, hanem egyenesen bátorítják. Végülis ezért hívják mobilnak. De mit csinál ilyenkor az IP? Az eredeti címmel már nem tud mit kezdeni a kedves felhasználó, mert az útvonalválasztók csakazértis az eredeti helyére küldik a csomagokat. Az útvonalválasztási táblázatok átírása kissé körülményesnek, és legfőképpen lassúnak tűnik, arról nem is beszélve, hogy mennyi sávszélességet kötne le teljesen feleslegesen. A megoldás az *(legalábbis egyelőre)*, hogy a vendéglátó ad új címet, mi meg hazaszólunk, hogy most éppen hol vagyunk elérhetők. Ezt a módszert mobil IP-nek hívják, és távolról sem ilyen egyszerű. Olyannyira nem, hogy erről még RFC sem született, és egyetértés hiányában egyik draft követi a másikat. A Microsoft természetesen ebben a küzdelemben is részt vesz, kicsit elmaradva ugyan a legfrissebb draftoktól, de a protokoll körüli zűrzavar fényében ezt igazán nem róhatjuk fel nekik. Aktualitása miatt legközelebb a mobil IPv6 protokollt és annak microsoftos vonatkozásait fogom bemutatni. Az érdeklődőknek addig is azt javaslom, hogy látogassák meg az **[1]** weboldalt, töltsék le, és próbálják ki a legújabb MSRIPv6-ot! Kellemes élményben lesz részük.

Zacco
zacco@fw.hu

A cikkben szereplő URL-ek:
[1] http://research.microsoft.com/msripv6/msripv6.htm

Következő számunk tartalmából

Windows 2000

A meghatalmazotti kapcsolat (trust) leleplezése.

Oknyomozó riportunkban annak járunk utána, hogy vajon a Windows 2000 Professional és Windows XP eltér-e a kiszolgálóváltozatoktól (Server és Advanced Server), és ha igen, miért nem. Bebizonyítjuk, hogy a munkaállomásváltozatok is képesek trust kapcsolat létrehozására – még ha a szakma nem is annak hívja e lépést.

XMLgessünk

XML.NET III.

Hogyan lehet egy relációs adatbázis adatai XML-ként látni és kezelni, transzformálni? Nem SQL Serveren. Hogyan lehet XML adatokat relációs adatként láttatni? Hogyan lehet ez minimális programozással megtenni? Ezekről beszélünk a következő részben, természetesen .NET alapon.

.NETAcademia

.NET alaptípusok II.

Folytatjuk a Common Type System feltérképezését az interfészek fogalmával, a közös ős, a System.Object szolgáltatásaival, valamint az érték- és referenciatípusok használatával.

ASP Suli:

Még mindig WAP

Miután nagyjából megismertük a WML nyelv felépítését és működését, nekikezdhetünk az első wapos weboldalunk létrehozásának. A következő számban bemutatjuk, hogyan kell ehhez az Internet Information Services beállításait módosítani, illetve áttekintjük a fejlesztéshez használható eszközöket, és azok használatát.

RIS:

Telepítőkészletek

CDROM alapú vagy RIPREP telepítőkészletet válasszunk? Előnyök, hátrányok, alkalmazási helyzetek. Mit kell és mit lehet megváltoztatni a SIF állományokban? Ezekre a kérdésekre keressük a választ a következő részben.

Farkasokkal táncoló

Szolgáltatások

Jobbról is, balról is körbejártuk már a fürtszolgáltatást, szinte minden beállítást ismerünk, csak még erőforrásokat nem hoztunk létre egy szálat sem. A következő fejezetet kizárólag e témának szenteljük.

SQL Server

A halálos ölelés

Az SQL Server belső felépítésének boncolgatása során elérkezünk a zárolási (lock) rendszer ismertetéséhez. Mi módon akadályozza meg az SQL Server, hogy párhuzamosan futó tranzakcióink felülírják egymás félkész adatát? Mi a teendőnk a zárolással kapcsolatban? Hogyan lehet elkerülni a legsúlyosabb esetet, a deadlockot?

Jogi esetek

Az Internettörvény

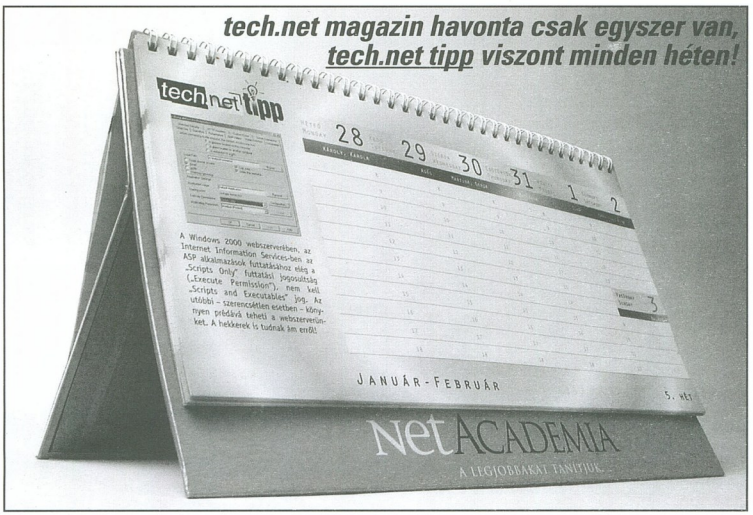
Internettörvény vagy csak az elektronikus kereskedelem szabályzása? Decemberben elfogadta az Országgyűlés a szakma által korábban csak Internettörvényként emlegetett elektronikus kereskedelmi törvényt. Karácsonyi ajándékként december 24.-én a Magyar Közlönyben meg is jelent a jogszabály szövege. Cikkünk bemutatja, hogy a sok szakmai vitát, egyeztetést megélt törvény mit és hogyan kíván regulázni?

Előfizetési szelvény

CÍMZETT: NETACADEMIA KFT.
FAXSZÁM: (1) 261-7145

FAX

Tisztelt Olvasónk!
Lapunk hírlapárusi forgalomba nem kerül, ezért ha kíváncsi megkezdett sorozataink folytatására, kérjük töltse ki és juttassa el hozzánk az alábbi megrendelőlapot. Előfizetőinket szeretettel várjuk Mesterkurzusainkon. Klubtagjaink kedvezményesen vehetnek részt konferenciáinkon, tanfolyamainkon stb.
Kérjük töltse ki ezt az előfizetési szelvényt és faxolja el az (1) 261-7145-ös faxszámra.



Minden előfizetőnk megkapta Magyarország egyetlen informatikai asztali naptárát, csak Önnek nincs! Ilyen körülmények között nem lehet normálisan dolgozni! Hisz nem jut hozzá a hét tippjéhez! Fizessen elő ***MOST,*** *mert 2002 első száz előfizetője még hozzájuthat a naptárhoz!*

Bugvadászat!
Vigyázat! A naptár bugos!
A hibák megtalálói ajándékot kapnak!

http://technet.netacademia.net/subs • e-mail: terjesztes@netacademia.net • fax: (1) 261-7145

Előfizetem a **tech.net** magazint: ...példányban egy évre (14.784 Ft)
...példányban fél évre (7.392 Ft)

Az előfizetés kezdete:...../.....

Előfizető neve:. . . . . .

Cég neve: . . . . . .

Cím: ☐☐☐☐ . . . . . .

E-mail cím: . . . . . .

Telefon: . . . . . .

Fax:. . . . . .

Fizetés módja: ☐ csekken *(postán küldjük)* ☐ átutalással

Kelt:...../...../..... Aláírás:.....

Amennyiben a számlázási cím nem egyezik meg a szállítási címmel, kérjük az alábbi részt is töltse ki!

Számlázási cím: **Szállítási cím:**

....

....

tech.net
working with windows

NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.