1.344 Ft

II. Évfolyam 7. szám

ISSN 1585185

tech.net

A Microsoft Magazinja

2001 Július

tech.net

A Microsoft Magyarország Szakmai Magazinja

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@lyris.netacademia.net

Kiadja és terjeszti
a **NetAcademia Kft.**
Terjesztési, előfizetési információ:
Tel.: 263-2732
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft
Példányszám: 3.000

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

Hirdetésfelvétel:
Bársonykalapács Marketing
Felelős: **Udvarev Rita**
Tel./Fax: 214-0923
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

Tech.Ed 2001 Barcelona

Talán meglepte kedves olvasóinkat a címlap, mely a barcelonai Tech.Ed konferencia jegyében fogant, ám ezzel egyidejűleg egyetlenegy cikk sem szól magáról az eseményről. Ennek oka egyszerű: a konferencia nemhogy nem zárult még le a lapzárta pillanatában, hanem jóformán el sem kezdődött. Összesen két bevezető előadáson és egy kiadós ebéden vagyunk túl, de ennyi is elegendőnek bizonyult a konferencia fő témájának megállapításához: .NET! Az augusztusi számban – reményeink szerint – közre tudjuk adni az első néhány Barcelonában fogant cikket.

A konferencia helyszíne a barcelonai olimpia és a világkiállítás céljára emelt Montjuic 2 épületegyüttes, melyben szinte elvész a 9000 résztvevő.

Néhány statisztikai adat: 9000 résztvevő, mintegy harminc országból. 35 Celsius fok, 20 perces várakozás reggelenként a konferenciabuszra. Magyarországról több, mint 160-an vesznek részt a rendezvényen. Kiosztva 18.000 darab Visual Studio .NET Beta 2. A .NET szolgáltatásokat huszonhat programozási nyelven lehet majd elérni. A számoknál azonban többet mond egy kis élménybeszámoló. E havi köszöntőnket rendhagyó módon két szerző jegyzi: Fülöp Miklós *(F.M.)* és Fóti Marcell *(F.M.)*.

F.M.: A konferencia egészére rányomja bélyegét, hogy a Microsoftnak ismét van mondanivalója. A tavalyi rendezvénnyel összehasonlítva ez a megállapítás különösen éles, hisz a rendezők akkor is kitettek magukért, de nem sok újdonsággal tudtak szolgálni. Miért? Mert tavaly egy céljait elért, megállapodott Microsoft állt szemben a hallgatósággal. Bill 1975-ös terve régesrég megvalósult, hisz minden íróasztalon évek óta ott a PC, Amerikában már otthonra is a sokadikat veszik a polgárok. Véget ért a böngésző, a Linux, a Novell, az SQL és a levelezőkiszolgáló háború. Bill valószínűleg nehéz döntés előtt állt: ha minden asztalon ott a PC, akkor akár nyugdíjba is mehet :-)

F.M.: Most azonban megint van követendő cél: a dotnet, mely valami hasonló változás, mint a DOS->Windows váltás. Akkoriban az egyalkalmazásos modellről tértünk át a több program párhuzamos, együttműködő futtatására. Ki emlékszik már arra, hogy anno nem volt vágólap? Papírral és ceruzával vittük át az adatokat egyik alkalmazásból a másikba. Ma is kell papír és ceruza: másképp nemigen tudjuk saját adatainkat átvinni egyik webkiszolgálóról a másikra. Ezen *(is)* változtat a .NET.

F.M.: A .NET több, mint amit elképzelünk.

- Új fejlesztési paradigma *(menedzselt kód, garbage collection stb. lásd múlt havi számunk e tárgyban született cikkét)*
- Új operációs rendszer, hihetetlenül sokat tudó új API-kkal *(ha hívhatjuk egyáltalán API-nak a .NET névtér objektumait)*
- Új, szabványos alapokra helyezett kapcsolattartás. Arccal az XML felé! Vesszen az RPC, itt az XML alapú SOAP!

Még meg sem tanultuk az LDAP-ot, LDIF-et, de már lassan nem is érdemes. Még az Active Directory is XML alapú lesz. A nem is távoli jövőben éppolyan könnyedén lehet majd XML kimenetű SQL lekérdezéseket futtatni az Active Directory-n, mint amilyen egyszerűen megtehető ez ma az SQL 2000-rel. Az XML nélkülözhetetlen abban a stratégiában, melynek célja az alkalmazások eljuttatása az összes eszközre.

F.M.: Tehát a .NET project keretében egyszerre, egyidőben kerül sor a Windows operációs rendszer objektumalapokra helyezésére, és a Windows, mint egyeduralkodó ügyféloldali rendszer kiváltására mobiltelefonokkal, hordozható ketyerékkel, hűtőszekrényekkel. Nem mellesleg az új programozási nyelv is születik: a C# *(a nyelvről részletes írás jelent meg lapunk tavaly szeptemberi számában, mely a weben teljes egészében elolvasható a* **[1]** *címen)*.

F.M.: Az informatikusok nyitottak a hülyeségre is. Mi sem bizonyítja ezt jobban, mint a Visual Basic.NET hatékonyságát igazoló bemutató. Példaképpen láthattuk a Bill Gates által 1981-ben *(vagy mikor?)* írott donkey.bas „játékprogramot", mely karakteres „grafikával" kápráztatta el az akkori fejlesztőket: autónkkal csacsikat kellett kerülgetni. Abban az időben a programsorok sorszámozásának megszüntetése volt a VisBas melletti fő fegyvertény. Majd bemutatták ugyanennek a játéknak VB.NET-tel készített változatát: real-time 3D grafika és hang, valósághű, cserélhető identitású webcsacsik, force-feedbackes kormányzás!

F.M.: Minden Tech.Ed elmaradhatatlan feltétele a számítógéphez, Internethez szokott hallgatóság ellátása az életfontosságú sávszélességgel. Ennek hagyományos módja az úgynevezett Communications Network, mely – immár hagyományosan – képtelen ellenállni a többezer informatikus rohamának, és időről időre beadja a kulcsot. Azonban ebben az évben ez nem sokakat zavar, mert drótnélküli hálózaton a teljes épületben magunkkal cipelhetjük az éltet adó sávszélességet. A Compaq vagy nyolcezer iPaq kézi-számítógéppel kedveskedett a jelenlévőknek, melyet ingyen és bérmentve használhatunk a konferencia teljes időtartama alatt.

F.M.: Ha már wireless, beszéljünk egy keveset a mobileszközök használatáról. Vajon mekkora erőfeszítésbe telik egy fejlesztőnek elkészíteni egy alkalmazás különböző eszközökre optimalizált változatát? A Visual Studio .NET használatával bizony nem sokba. Az ASP.NET például automatikusan böngészőre *(telefonkijelzőre)* optimalizálja a webes objektumokat...

F.M.: A Microsoft azt is célul tűzte ki, hogy a hagyományosan Microsoft-vonalon dolgozó 8 millió fejlesztőn kívül a .NET-et elérhetővé tegye gyakorlatilag az összes élő programozó számára: VB, VBScript, C, C++, C#, Java, Perl, Python, Rexx, Cobol *(!)*, Fortran *(!)*, Eiffel és még ki tudja milyen ismeretlen programozási nyelven lehet majd – állítólag – egyenértékűen dolgozni!

A mai nap programjából már csak az esti Country Drinks van hátra, elbúcsúzunk. Ígérjük, hamarosan fényképekkel illusztrált élménybeszámolót tárunk kedves olvasóink elé a **[2]** címen.

Fóti Marcell és Fülöp Miklós

A cikkben szereplő URL-ek:
[1] http://technet.netacademia.net
[2] http://technet.netacademia.net/teched2001

Office XP!

Itt az Office XP!

Megjelent az Office XP, a Microsoft irodai kiszolgálócsomagjának legújabb verziója. Az **[1]** címen részletesen olvashatunk a termék újdonságairól és előnyeiről.

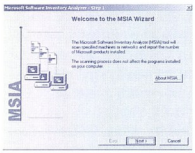
Megjelent az MSIA

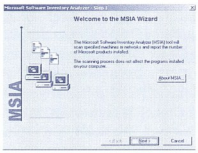
Az új szoftverlicencelésre történő lassú átállás, valamint a kisvállalatok szoftvergazdálkodásának megkönnyítése jegyében 2001 júliusában megjelent a Microsoft Inventory Analyzer nevű programocska, mely önálló alkalmazásként futva *(nem igényel sem SMS-t, sem SQL Servert)* lehetővé teszi, hogy akár a hálózat összes gépén felmérjük a telepített *(egyelőre csak MS)* szoftvereket. A 3,36 MB-os programocska az **[2]** címről tölthető le. Ez a régen várt eszköz rengeteg rendszergazdának oldja meg mindennapi gondjait: vajon hány Word, Excel és ilyen/olyan operációs rendszer van telepítve a hálózatban?

Talán már megszokhattuk, hogy jelentős számú Microsoft szoftver kis, külső cégnél kezdte pályafutását *(NLBS, Frontpage, Excel stb.)*, az MSIA mégis egyedülálló a maga nemében: Indiából származik **[3]**! Fontos megemlíteni, hogy a Microsoft folyamatosan frissíti, és az **[2]** címen elérhetővé teszi a felismerési adatbázist, mely már jelenleg is igen jelentős készlettel rendelkezik. A futás eredményéről Excel, HTML, de akár TXT formátumban kaphatunk jelentést.

Lapzárta után: Exchange 2000 Server Service Pack 1

Lapzárta után érkezett a hír, hogy megjelent végre a régóta várt első javítócsomag az Exchange 2000 Server-hez. Az SP1 a [4] címről tölthető le, és a hibajavítások mellett számos újdonságot is tartalmaz. Az AntiVirus API-k támogatása mellett *(amiről a 13. oldaltól részletesebben is szólunk)* több ponton továbbfejlesztették a webes felületet *(OWA)* – például a magyar nyelv támogatását; új az Exchange Server 5.5 Migration Tool, vagy éppen az ugyacsak Exchange 5.5-ből ismerős Mailbox Manager. Kötelező javítás!

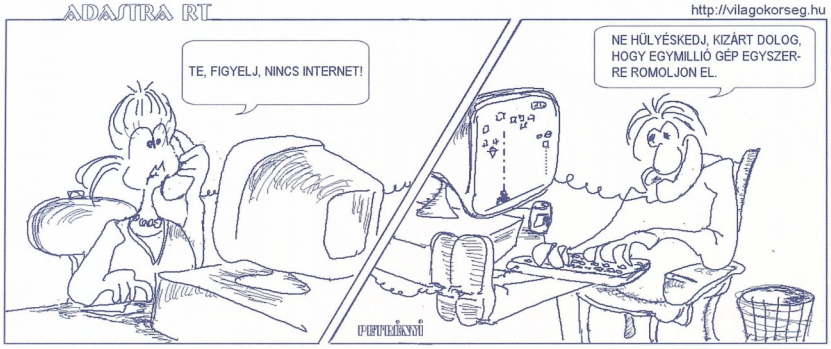




Az MSIA varázslóval akár kiszolgálótermékeket is kereshetünk a hálózatban

A cikkben szereplő URL-ek:

[1] http://www.microsoft.com/hun/product/OfficeXP.asp
[2] http://www.microsoft.com/piracy/msia
[3] http://www.aditi.com/
[4] http://www.microsoft.com/exchange/downloads/2000/sp1.asp



Network Monitor (VII. rész)

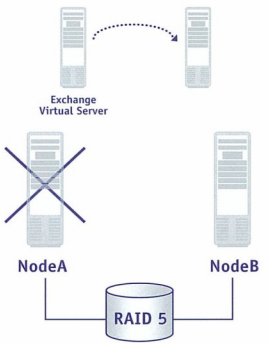
Az elmúlt alkalommal megígértem, hogy most a hitelesítések hálózati forgalmát vesszük sorra. Ígéret szép szó, nem tartom – úgy jó. Nem tartom a szavam, mert közben lezajlott egy NetAcademia tanfolyam a Windows 2000 fürttechnológiákról, s ezen készítettem néhány izgalmas pillanatfelvételt az NLBS *(Network Load Balancing Service)* működéséről, melyek értéke vetekszik egynéhány ritka műkincs, például a Mona Lisa, vagy egy ritka bélyeg értékével. Ezt osztom meg önökkel ebben a cikkben, gazdagodjunk mindannyian!

Fürt és fürt

A Windows 2000 sokféle beépített fürtözési lehetőséggel rendelkezik. Van megoldás nagy rendelkezésre állású rendszer kiépítésére *(Shared SCSI, vagy Failover Cluster)* éppúgy, mint terheléselosztás használatára akár 32 gép között *(NLBS)*. Ezekről ejtsünk pár szót, ha már volt szerencsém heteket tölteni a technológiákkal, s mind a könyökömön, mind a fülemen ez jön.

Failover Cluster

Osztott SCSI tárolón alapuló, vagy képletesebben „váltott lovakkal" fürt. Ez a fürtmegoldás – tipikus kiépítésben - két számítógép *(Node)* egyidejű működtetésével biztosítja, hogy a rajta futó szolgáltatások mindig elérhetőek legyenek, akár még az egyik gép teljes kiesése esetén is. Ebben a modellben központi szerep jut a két *(vagy Datacenter Server esetén több)* Node között elhelyezkedő osztott tároló alrendszernek, amelynek lemezeit a fürt végpontjai közösen használják. Ezek felett, úgynevezett virtuális kiszolgálókként, mindegy a kibertérben futnak az alkalmazások. Vagy méginkább: a Node a ló, a virtuális kiszolgáló a lovas, mely egy adott pillanatban egyszerre egy gép hátán ül, azt hajtja, sarkantyúzza. Ha a gazdagép rendetlenkedik, vagy leáll *(a ló megdöglik)*, a hűtlen virtuális kiszolgáló szedi a sátorfáját, és IP címestül, mindenestül átül a másik, hibátlan Node-ra *(Failover)*. Akárhol ül is a lovas, az adatokat mindig eléri, hisz azokat nem a lovon tárolja, hanem az istállóban: az osztott elérésű RAID alrendszeren.



Server Cluster Failover: NodeA kiesése esetén a rajta futó virtuális kiszolgálók átköltöznek a hibátlan NodeB-re

A Failover Cluster egyetlen különleges hálózati forgalma a két Node között zajló úgynevezett szívverés *(Heartbeat)*, melynek értelmezéséhez sajnos a NetMon nem sokat tesz hozzá: nincs ilyen parser benne.

NLBS terheléselosztó fürt

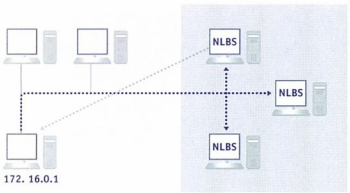
Az NLBS gyökeresen más elveken működik, mint az előbb említett Failover Cluster. Míg az előző fürtnél igen szigorú szoftver- és hardverkövetelményeknek kellett teljesülniük *(osztott SCSI alrendszer, maximum 2 Node Advanced Server esetén, úgynevezett Cluster Aware (fürtképes) alkalmazások stb.)* addig az NLBS-hez semmi nem kell. A Node-ok száma is jelentősen több lehet: maximum 32. Valójában az NLBS, bár rendszerkomponens, idegen testként ékelődik be a Windows hálózati architektúrájába, és mindenkit ott csap be, ahol tud *(mimikri)*. Még az NLBS-t futtató gazda operációs rendszer is csak „les", hogy mi történik, a rajta futó szolgáltatások ugyanis csak akkor kapnak adatot a hálózatról, ha ezt az NLBS eszközmeghajtó jónak látja. Mivel maga az oprendszer is csak „les", nem meglepő, hogy az NLBS-en fürtben futtatott alkalmazásoknak sincs halvány lila elképzelésük arról, hogy fürtben vannak. Egy IIS-nek, vagy egy Terminal Services-nek sejtelme sincs arról, hogy ők valójában – mondjuk - 32-en vannak!

A mimikri üzemmód az NLBS történelmi múltjából eredő feature, ugyanis nem Microsoft rendszerkomponensként kezdte az életét, hanem Convoy Cluster Software néven egy pici bété forgalmazta még az NT4-hez. Ez a zseniális kis cég azután felvásárlás áldozatául esett, így született előbb a WLBS *(A Convoy NT4-es elnevezése)* majd az NLBS *(a Convoy Windows 2000-es neve)*.

Az egész fesztivált a 63 kilobájtos *(!)* WLBS.SYS csinálja, mely minden tagkiszolgálón közvetlenül a hálózati kártya meghajtója felett, de a TCP és UDP réteg alatt beékelődik a rétegzett hálózati architektúrába, és elvégzi a szükséges műveleteket ahhoz, hogy a fürt tagjai egységesen viselkedjenek a munkaállomások felé.

Az NLBS működése

Az NLBS tisztán TCP/IP trükkökkel dolgozik. A fürtbe kötött számítógépek az NLBS üzembe helyezésekor közös *(!)* IP címet kapnak *(A régi egyedi címüket is megtarthatják persze. Már az is a WLBS.SYS okosságának tudható be, hogy nem kezdenek sírni a gépek IP cím ütközés miatt)*. Tehát lesz mondjuk 32 gépünk, közös IP címmel. Ebből kitalálható, hogy a munkaállomások kérései a fürt összes tagjához befutnak. A WLBS.SYS-ek pedig szépen eldöntik, hogy a sok Node közül melyik vegye magára a kérdést, melyik válaszoljon.



Az NLBS fürtben - a közös IP cím miatt - mindegyik tag megkapja a felhasználó kérését, de csak egyikük válaszol

Ez roppant egyszerűen hangzik, de a valóságban nem az. Ugyanis nincs idő arra, hogy a WLBS.SYS-ek a kérés beérkezésekor nekiálljanak egyeztetni a többi ikertestvérrel, hogy melyikük érne rá jobban a feladat kiszolgálására. Ha az egyeztetés az igény beérkezésekor történne, odalenne a teljesítménynövekedés. A megvalósított szellemes megoldás ennél sokkal nagyobb teljesítményt biztosít:

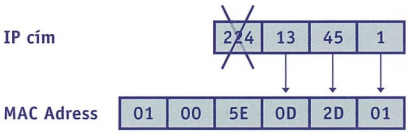
1. A fürt tagjainak beállításakor, vagy a fürtszabályok módosításakor a tagok gyorsan előre leosztják egymás között a lapokat, azaz jó előre megállapodnak, hogy a világ minden sarkából érkező kérésekre melyikük fog reagálni *(a lapok leosztását a fürt konvergálásának nevezik hivatalosan, s erről a tényről bejegyzés készül az Event Logba)*.
2. A kérés beérkezésekor – amely a közös IP cím miatt mindegyik taghoz eljut – mindegyik gép önállóan eldönti, hogy a leosztott lapok alapján vajon övé-e a feladat. Ha igen – a WLBS.SYS továbbengedi a kérést az operendszernek, s ez a gép válaszol. Ha nem – a WLBS.SYS eldobja a csomagot.

Azt gondolhatnánk, hogy ezzel vége is minden load balance lehetőségnek, dinamikus terheléseloszlásnak, hisz ha az előre leosztott zsugák között nincs ott egy lap, akkor nincs ott. Ám valójában a konvergencia során nem IP tartományokat osztanak el egymás között, hanem mindegyik tag közös szabályokhoz jut a saját DINAMIKUS döntésének meghozatalához! Vagyis a Node-ok összebeszélnek, és azután ahhoz tartják magukat. Megbeszélik a terheléselosztás szabályait *(portok, címek, arányok, affinity stb.)*, majd ezzel felszerelkezve egyedileg, egymástól függetlenül döntenek egy-egy kérés megválaszolásáról – és sosem akadnak egymással össze! Zseniális!

A beállítható terhelési szabályokat most nem elemezném, hisz a Network Monitor miatt gyűltünk itt össze, így lassan rátérünk a lényegre. Ehhez meg kell ismerkednünk azzal, hogy vajon hogyan képesek a gépek közös IP címre reagálni. Úgyvan. Közös MAC addressel! Két módon érhető el közös MAC Address használata az NLBS alatt:

- Unicast módban a kártyák elveszítik gyári eredeti MAC Addressüket *(Hoppá!!)*, mert az NLBS mindegyik Node gépen azonos MAC Addresst vés a kártyákra. *(A vésés túl erős kifejezés: reboot után visszajön a gyári MAC)*. Az eredeti MAC Address elveszítésével ezek a gépek többé a *<?$"@ életben nem lesznek külön-külön megcímezhetők, így jó előre tessenek felmásolni rájuk a közös weblapokat, mert később már ezen a kártyán keresztül nem lehet! Sőt, távolról egyáltalán nem fogjuk tudni megkülönböztetni a gépeket, egyszerre csak egyikük fog válaszolni - hisz erre való a fürt. Ez az üzemmód eléggé értelmetlen.
- Multicast módban a kártyák megtartják eredeti MAC Addressüket, és ezzel a gépek is megtartják egyedi elérhetőségüket, menedzselhetőségüket, s a meglévő, gyári eredeti MAC Address ***mellé*** kerül fel egy Multicast MAC, melyre a fürt tagjai „harapni" fognak. Ez az üzemmód a tökéletes választás, ezen a nyomon haladunk tovább.

A Multicast MAC Address képzésének szabványos formája a következő: vedd a kártyához rendelt Multicast IP címet, és annak utolsó három bájtját tedd be a MAC Address utolsó három bájtjába, az első három bájt pedig legyen 01-00-5E. Valahogy így:



A Multicast MAC Address boszorkánykonyhája

(A Multicastról a BYTE magazin júniusi számában megjelent cikkem értekezik ennél részletesebben, ezt az ábrát is onnan loptam vissza.)

Ezek után már csak azt kell biztosítani, hogy a munkaállomások ARP kéréseire ez a Multicast MAC jöjjön vissza.

Szabványok határán

De megint bökkenőbe ütközünk, ugyanis itt nem valódi Multicastról van szó, ahol egy adó kiszolgál egy halmaz vevőt, hanem ál-Multicast, ami valójában Unicast – legalábbis a fürtöt használó munkaállomások szemszögéből. Ez a gond igazán nagy gond, ugyanis az eredeti Multicast szabványban **[2]** szó sincs ARP-ról; a Multicast csoporthoz csatlakozni az IGMP protokoll segítségével lehet. Ráadásul Multicast IP címekkel *(D osztály, 224.0.0.0-239.255.255.255)* sok végpont –beleértve a Windows operációs rendszereket nem lenne hajlandó Unicast forgalmat kezdeményezni.

Emiatt az NLBS trükkhöz folyamodik: ha már ál-Multicast, legyen teljesen spéci! Ezért a fürt közös címe NE Multicast, hanem rendes, kutyaközönséges IP cím legyen, innentől a munkaállomások valóban ARP-vel fognak hozzá MAC Addresst keresni. A Multicast MAC Address sem a szabványban leírt 01-00-5E kezdetű lesz, hanem 03-BF, amely még mindig csoportos cím ugyan, de lokális *(MAC Addressből is van lokális tartomány, úgy kell használni, mint IP-nél a belső címeket! Bizony!)*

Nézzük meg e varázscímeket a Network Monitorral! E hónapban az **[1]** címről az NLBS.CAP fájl tölthető le, amelyben egy munkaállomás megpingel egy NLBS fürtöt. Pár oldal múlva ebből ki tudjuk majd olvasni, hogy a fürtnek hány csomópontja volt. A fájlnak vizsgáljuk most meg a harmadik csomagját *(ICMP Echo)*, keressük meg a címzett MAC Addressét. Eddig még egyszer sem boncolgattuk magát a MAC Addresst, úgy tekintettük, mint monolitikus, felbonthatatlan számot. Most azonban kattintsunk rá kettőt, bontsuk ki! Ezt kell látni:

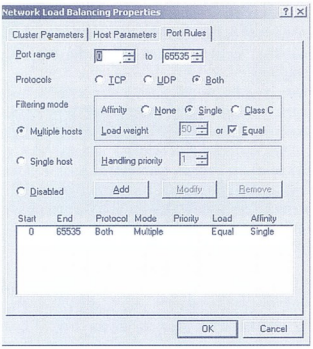
```
ETHERNET: Destination address : 03BFC0A8040A
ETHERNET: .......1 = Group address
ETHERNET: ......1. = Locally administered address
```

A legelső bájt alsó bitje mutatja, hogy ez egy csoportos MAC Address *(Group Address)*, a második bit pedig arról árulkodik, hogy ez egy belső, lokális MAC Address.

Ha pedig decimálissá alakítjuk a számsorozatot *(3-191-192-168-4-10)* a NetAcademia tanfolyamok hallgatói fel fogják ismerni az egyik tanteremben használatos IP címet a MAC Address utolsó négy bájtján *(192.168.4.10)*.

NLBS portszabályok

A következő ábrán egy pillantás erejéig megtekintjük az NLBS fő párbeszédpaneljét annak érdekében, hogy átfogó képet kapjunk az itt beállítható szabályok felépítéséről.



Az NLBS portszabályai

Jól látható, hogy a WLBS.SYS terheléselosztási és egyéb szabályai, szűrései mind TCP, mind UDP portokra beállíthatók – másra azonban nem! A fenti ábra tanúsága szerint ha a fürt csomópontjára beérkező kérés nem TCP és nem is UDP, akkor ez nem esik az NLBS fennhatósága alá! Jól jegyezzük meg: az NLBS csak és kizárólag a TCP és UDP protokollok forgalmát fürtösíti. Vajon mit tesz a WLBS.SYS a szabályok alól „kilógó" csomagokkal *(például ARP-vel, ICMP-vel, IGMP-vel, GRE-vel stb.)*? Nem dobhatja el, hisz ezek nélkül nincs élet. Tehát átengedi, felküldi az operációs rendszernek, amely válaszol rá. Lássuk az NLBS.CAP fájlban az ARP protokollt! Szűrjük meg az NLBS.CAP fájlt, hogy csak ARP protokoll maradjon a szemünk előtt. Hány csomagra számítunk, ha elárulom, hogy a felvétel zajmentes, tiszta környezetben készült? Én kettőre számítanék: egy ARP kérésre és egy válaszra. Ám nekünk három ARP csomagunk van...

ARP az NLBS fürtben

A vak is láthatja: egy ARP kérdésre két válasz érkezett! Ahogy a fenti okfejtésből talán következtethettünk rá: a fürt összes csomópontja *(mind a kettő :)* válaszolt az ARP kérésre, mégpedig az alábbi ábra szerint nem a fent levezetett közös MAC Addressel, hanem a hálókártyák saját, beépített fizikai címével *(0002A5095EB8 és 0002A509618B)*!

```
1.742505  0090272CE129  *BROADCAST    ARP_RARP  ARP: Request, Target
1.742505  0002A5095EB8  0090272CE129  ARP_RARP  ARP: Reply, Target IP
1.742505  0002A509618B  0090272CE129  ARP_RARP  ARP: Reply, Target IP
```

Egy ARP kérdésre két válasz? És két különböző címről?

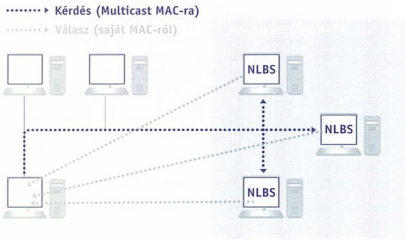
Ajaj! Mi lesz itt?
Vajon mi kerül szerencsétlen munkaállomás ARP cachébe? Az a MAC Address, amelyik előbb válaszolt *(mert az győz)*? Vagy amelyik később *(mert felülüti a korábbi választ)*? De az tisztán látszik, hogy a beígért Multicast MAC Addressnek nyoma sincs. Hacsak...
Hacsak ki nem bontjuk ezt az átkozott ARP Reply-t!

```
ETHERNET: ETYPE = 0x0806 : Protocol = ARP:
  Address Resolution Protocol
  ETHERNET: Destination address : 0090272CE129
  ETHERNET: Source address : 0002A5095EB8          ← 1
  ETHERNET: Frame Length : 60 (0x003C)
  ETHERNET: Ethernet Type : 0x0806 (ARP:  Address
  Resolution Protocol)
  ETHERNET: Ethernet Data: Number of data bytes
  remaining = 46 (0x002E)
ARP_RARP: ARP: Reply, Target IP: 192.168.4.200
Target Hdwr Addr: 0090272CE129
  ARP_RARP: Hardware Type = Ethernet (10Mb)
  ARP_RARP: Protocol Type = 2048 (0x800)
  ARP_RARP: Hardware Address Length = 6 (0x6)
  ARP_RARP: Protocol Address Length = 4 (0x4)
  ARP_RARP: Opcode = Reply                  ② 2
  ARP_RARP: Sender's Hardware Address =
  03BFC0A8040A
  ARP_RARP: Sender's Protocol Address =
  192.168.4.10
  ARP_RARP: Target's Hardware Address =
  0090272CE129
  ARP_RARP: Target's Protocol Address =
  192.168.4.200
  ARP_RARP: Frame Padding
```

Ha összevetjük az ① helyen olvasható MAC Addresst, *(amely a feladó címe)* a ② ponton látható címmel *(mely az ARP válasz tartalma szerinti feladócím)* azt tapaszthatjuk, hogy ez az ARP csomag kissé „hibás": látszólag nem attól a géptől jön, mint aki magának érezte a kérdéses IP címet! *(És ez a csoda azután mégegyszer befut szerencsétlen munkaállomáshoz a fürt másik csomópontjáról is: győzze feldolgozni!)* Amíg ugyanis Ethernet szinten a hálókártya eredeti MAC Addresse szerepel, addig a beágyazott ARP-ben már az a Multicast katyvasz van, amit egy oldallal korábban elemeztünk! A fürt tagjai Ethernetszinten letagadják a Mulitcast jelenlétét, míg a beágyazott ARP-ban már ott virít a Multicast MAC!
Miért, oh miért?
S ha a kezdeti kapcsolatfelvételkor használt ARP ilyen „letagadós", akkor vajon a többi hálózati forgalom milyen lehet?

Fürtök és switchek

A válasz nem a Microsoft háza táján keresendő. Bizonyos hálózati elemek MAC-kezelési furfangjai miatt van szükség a Multicast MAC elrejtésére a válaszcsomagokban. Hogyan is működik egy switch? Virtuális pont-pont kapcsolatot alakít ki a kommunikáló felek között arra az időre, amíg a csomag átkerül a címzettől a feladóhoz. A switch a kommunikáló felek közt „félúton" állva nulla milliszekundum alatt képes eldönteni, hogy melyik két portját kell összekötnie egy adott csomag elszállításához. Erre a rendkívül gyors cselekvésre azért képes, mert minden portjához megtanulja az azon lógó gép MAC Addresését – egyszerűen kiolvassa és megjegyzi a rajta áthaladó forgalomból. Sajnos a switcheket nem készítették fel NLBS-re, nem igazán van olyan szabály bennük, hogy egyszerre X portot kössenek össze. Még meg is bolondulhat némelyikük, ha több portján egyidőben ugyanaz a MAC Address bukkan fel. Az a biztos, ha nem hagyjuk, hogy „megtanulja" a Multicast MAC-et, hanem eldugjuk előle. Bújócska. Ennek záloga, hogy válaszcsomag Ethernet fejlécében sohase jelenjen meg a Multicast MAC, hogy switch pajtás ne tudja onnan kiolvasni és memorizálni. Emiatt válaszolnak a közös MAC Addressű fürtvégpontok a saját MAC-kel!



A feladó fürttagok sosem a Multicast címmel válaszolnak

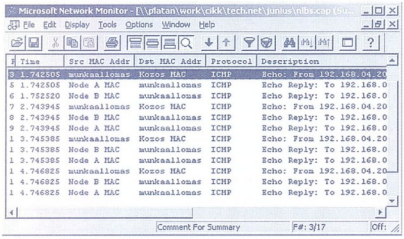
Ha valaki hubot használ, vagy turbóokos switch áll rendelkezésére, a bújócskát ki is kapcsolhatja a

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
Services\WLBS\Parameters\MaskSourceMAC
```

érték nullázásával. Én nem tenném...

PING a fürtre!

A munkaállomás a fenti ARP Reply csomagokból kihalászott Multicast MAC címet használatba véve ICMP Echo-ba kezd. Lássuk a fürt pingelését! A fenti MaskSourceMAC bekapcsolt állapotában zajló ICMP forgalom könnyebb értelmezése érdekében a MAC Addresseket elnevezésekkel láttam el.



Egy ICMP Echora két válasz?

Az ábra lényege így foglalható össze:

```
Munkaallomas -> ICMP Echo  -> Kozos MAC
Munkaallomas <- ICMP Reply <- Node B MAC
Munkaallomas <- ICMP Reply <- Node A MAC
```

A pingelő parancssorban egyébként csak egyetlen válasz látszik, kizárólag a NetMon számára tárul fel a valóság.

Furcsa példáim láttán esetleg egyesekben az a kép ragad meg, hogy az NLBS fürt használhatatlan, mert mindig minden csomópont válaszol a munkaállomások kérdéseire, s ezzel eldugítja a hálózatot. Ne tévedjünk, eddigi példáim nem az NLBS üzemszerű működésére vonatkoznak! A TCP és UDP fölött futó alkalmazások *(például IIS, Terminal Services stb.)* természetesen nem ezt a multiválaszolós hálózati forgalmat mutatják, hisz azokra hatnak a portszabályok, és a WLBS.SYS csak egyetlen fürttag válaszolását teszi lehetővé.

Az NLBS korlátai

Most, hogy „mindent" tudunk az NLBS viselt dolgairól összefoglalnám a működtetés főbb kereteit:

- Nem használható ez a fürttípus olyan környezetben, ahol valamelyik hálózati elem *(például útválasztó)* nem viseli el az ál-Multicastot *(rendes IP címhez Multicast MAC)*.
- Nem lesz elérhető az NLBS fürt akkor sem, ha valamelyik hálózati elem nem tolerálja azt a működésmódot, hogy nem arról a MAC Addressről érkezik a válasz, ahová a kérdés irányult *(ilyenkor kipróbálhatjuk a MaskSourceMAC kikapcsolását)*.
- Legkönnyebben read only alkalmazások *(IIS, Terminal Services, Proxy)* futtatására használható, mivel ilyenkor nem kell gondoskodni a fürt csomópontjain szétszórtan felbukkanó adatok összesítéséről.

AppCenter Server

A fürt beüzemeléséről egy szót sem ejtettem, s nem véletlenül. Nem egyszerű ugyanis puszta kézzel kialakítani 32 azonos paraméterezésű csomóponti gépet. Nem beszélve a változások követéséről! Megállapítottuk, hogy az alkalmazások *(például IIS)* nem tudják, hogy fürtön futnak. Emiatt nem is képesek a megváltozott adatokat *(például ASP lapokat, ActiveX vezérlőket stb.)* szinkronizálni. A hamarosan megjelenő AppCenter Server a fürtök körül felmerülő napi feladatok ellátásában nyújt hathatós segítséget. Két kattintással elvégezhetővé válik új fürtcsomópontok teljeskörű, alkalmazásszinkronizációs telepítése. Ezzel a termékkel külön cikkben fogunk foglalkozni.

Fóti Marcell
marcellf@netacademia.net
MCSE, MCT, MCDBA

A cikkben szereplő URL-ek:

[1] http://technet.netacademia.net/download/netmon
[2] RFC 1112 Host Extensions for IP Multicasting
http://www.ietf.org/rfc/rfc1112.txt?number=1112
[3] RFC 1700 Assigned Numbers
http://www.ietf.org/rfc/rfc1700.txt?number=1700

A System Attendant Service az Exchange 2000-ben

A Microsoft Exchange 2000 Serverben már a neve alapján is el tudjuk képzelni, hogy mit csinál az Information Store szolgáltatás, illetve azt, hogy mire jó az egyes protokollokat meghajtó Internet Information Services komponens, de vajon tudjuk-e pontosan, hogy mit is csinál a System Attendant? Ebben a cikkben megpróbálom összefoglalni a Sysem Attendant Service legfontosabb funkcióit, illetve, hogy kell-e rajta bármit is hangolni, és ha kell, akkor azt hol és hogyan tehetjük meg.

Mit is csinál a System Attendant?

A System Attendant *(SA)* több egymástól eltérő funkciót megvalósító szálat futtat magában. Ezek a szálak nem külön processzként futnak, ezért nem látjuk ezeket a Services applet, vagy a Task Manager futtatásakor a futó szolgáltatások, processzek között. Ennek teljesítményszempontból van szerepe, a szálak közti kontextusváltás sokkal kevesebb erőforrást igényel, mint a processzek közti váltás. Viszont az eseménynaplóban és a különböző információforrásokban gyakran úgy találkozunk ezekkel a SA belsejében futó szálakkal, mintha külön szolgáltatások lennének. Ezért fontos pontosabban megismernünk a System Attendantot.

Nevesített szolgáltatásként futó szolgáltatások:

- Metabase Update Service - IIS metabase és AD közötti replikáció
- Recipient Update Service – a levelezésbe bevont címtárobjektumok különböző címlistákhoz rendelését végzi
- DSAccess Service – az Exchange 2000 AD elérését biztosítja
- DSProxy Service – régebbi MAPI kliensek címlistaelérését biztosítja

Nem nevesített szolgáltatásként végrehajtott SA funkciók:

- Levelezésbe bevont nyilvános mappákhoz tartozó címtárobjektumok létrehozása és törlése
- Emeltszintű biztonság bekapcsolásakor a bizonyítványok továbbítása a felhasználók részére
- Levelesláda-kvóták figyelése, figyelmeztető üzenetek és tiltások generálása
- Törölt elemek kétfázisú eltávolítása
- Message Tracking log kezelése

Metabase Update Service *(MUS)*

Az Exchange 2000 szorosan integrálódik a Windows 2000 alapszolgáltatásai között nyilvántartott Internet Information Services 5.0 *(IIS)* komponensbe. Az Exchange 2000 telepítésekor frissíti az IIS meglevő protokollmeghajtóit *(SMTP, NNTP)*, és újakat is telepít *(POP3, IMAP4)*. Az IIS egy külön tárolóban tartja konfigurációs adatait, az úgynevezett metabase-ben *(lásd tech.net magazin 2001. január, 16. oldal)*. Ez egy memóriában tárolódó adatbázis, mely nagyon gyors adatelérést eredményez, ami kritikus a különböző protokollok nagyteljesítményű futtatásához. *(A metabase.bin állomány adja ennek az adatbázisnak a kezdőértékeit rendszerinduláskor, illetve ide íródik ki az adatbázis leálláskor.)* Az Exchange 2000 ezzel szemben az Active Directory konfigurációs partíciójában tárolja a beállításait. Annak érdekében, hogy az IIS továbbra is a számára optimalizált metabase-ből tudjon dolgozni és ne kelljen az Active Directory-hoz fordulnia, szükséges az AD és a metabase közti adatszinkronizáció.

Ezt a feladatot látja el a Metabase Update Service *(MUS)*, mely kiolvassa a konfigurációs adatokat az Active Directory-ból és beírja az Internet Information Services metabase-ébe. A Metabase Update Service tehát nem külön szolgáltatás, hanem a System Attendant Service alatt fut, fogadja az Active Directory címtárváltozási értesítéseit, melyek az IIS keretein belül futó protokollokkal kapcsolatos konfigurációs változások hatására generálódnak. Ezen értesítések alapján automatikusan frissíti az IIS metabase-t.

A fő jelentősége ennek a szolgáltatásnak, hogy elég bármely tartományvezérlőhöz csatlakoznunk az Exchange System Managerrel, és az ott végrehajtott konfigurációs változtatások automatikusan érvényre jutnak a megfelelő Exchange 2000 kiszolgálón ezen szolgáltatásnak köszönhetően, még akkor is, ha az adott kiszolgálóhoz nincs is állandó hálózati kapcsolatunk. Persze ahhoz hogy ez jól működjön, az Active Directory replikációnak megfelelően kell működnie, illetve a MUS-nak el kell tudnia érni egy tartományvezérlőt.

A szolgáltatás működéséből adódóan az is kiderül számunkra, hogy ha direktben módosítjuk a metabase-t, akkor az felülíródik az Active Directory-ban tárolt információkkal, mert a Metabase Update Service az adatokat csak egy irányba, az Active Directory-tól a metabase felé frissíti.

Vajon miért kell erről a komponensről tudnunk? Egyrészt azért, mert tevékenységéről az eseménynaplóban bejegyzéseket találunk. Mindazon esemény, melynek forrása az MSExchangeMU, az a Metabase Update Service működésének *(vagy hibájának)* a következménye. Másrészt az Exchange 2000 rendszerünk finomhangolásakor jól jön a komponens működésének nyomonkövetése. Például, amikor a fájlrendszerbeli SMTP Mailroot könyvtár tartalmát szeretnénk a rendszerpartícióról egy másik meghajtóra tenni. Ennek a könyvtárnak az áthelyezésére nincs grafikus felület az Exchange System Managerben, még csak ne is a Metabase Editor eszközzel kísérletezzünk! Akkor mit is tudunk tenni a Default Virtual SMTP Server mailroot alatt használt mappáinak áthelyezése érdekében?

Állítsuk le az Exchange szolgáltatásait, többek között a IIS-t is. Mozgassuk el a kívánt új helyre az Exchsrvr\mailroot mappa alól a megfelelő *(jelen esetben a VSI 1)* mappát és összes almappáját. Vegyük elő az ADSIEdit eszközt, és keressük meg a konfigurációs konténer következő objektumát:

```
Configuration Container\ CN=Configuration,
CN=Services,
CN=Microsoft Exchange, CN=<szervezet>,
CN=Administrative Groups,
CN=<admin csoport>,
CN=Servers,
CN=<kiszolgáló>,
CN=Protocols,
CN=SMTP,
CN=1
```

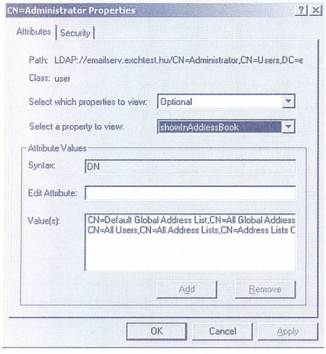
(jelen esetben a Default Virtual SMTP Server beállításait módosítjuk).
Ennek az objektumnak keressük ki a következő attribútumait és írjuk át a kívánt új elérési útnak megfelelően:

```
msExchSmtpBadMailDirectorys
msExchSmtpPickupDirectory
msExchSmtpQueueDirectory
```

Amennyiben több tartományvezérlőnk van, várjuk meg, hogy a címtárreplikáció végrehajtódjon. Indítsuk el a System Attendant szolgáltatást *(benne elindul a Metabase Update Service)*, és várjuk meg a sikeres metabase frissítést, melyet az eseménynaplóban a 1005-ös események jeleznek *(forrás: MSExchangeMU, kategória: General)*.
Ezzel a címtárbeli adatoknak megfelelően frissült a metabase, indíthatjuk a további Exchange szolgáltatásokat, és az SMTP már az új könyvtárakban fog dolgozni.

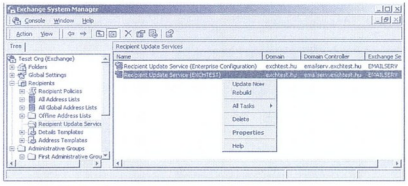
Recipient Update Service *(RUS)*

Az Exchange 2000-nek nincs saját címtára, hanem a Windows 2000 Active Directory-ját használja erre a célra. Azaz az Active Directory-ban található felhasználói és csoportobjektumok használhatók fel címzésre is. Vajon akkor mit kell itt frissíteni egy ilyen szolgáltatással? Az Exchange 2000 egyszerre több címlistát is tud kezelni. Az egyes címlistákhoz tartozás feltételét egy-egy LDAP lekérdezés határozza meg. A levelek címzésére felhasználható objektumoknak van egy showInAddressBook tulajdonsága, mely egy „multivalue" mező, és ide íródik be mindannak a címlistának a neve és helye, amelyeknek az adott objektum tagja.



Címlista tulajdonság

A másik attribútum – amit ez a szolgáltatás frissít - az objektum e-mail címeit írja le. Ezek az e-mail címek leggyakrabban egy Recipient Policy-ban definiált LDAP lekérdezés és címminta alapján generálódnak a RUS szolgáltatás által.
Amikor egy új tartományba telepítjük az Exchange 2000-et, vagy olyan tartomány felhasználóinak is akarunk címezni leveleket, amelyben nincs Exchange 2000, akkor szükségünk van a Recipient Update Service-re egyrészt a levelezésbe bevont objektumok címlistához rendelése, másrészt az e-mail címek generálása miatt.
A RUS két módban működik: az egyik a tartomány névterében *(Domain RUS)*, a másik az AD konfigurációs névterében *(Enterprise RUS)*. A tartománynévtérben a felhasználók és csoportok vannak, így a Domain RUS az ezekkel kapcsolatos címlistaattribútumok frissítéséért felel. Az Enterprise RUS a Store adatbázisért, magáért a System Attendant Service-ért, az MTA-ért felel, melyeknek szintén van e-mail címük, hogy a különböző Exchange 2000 kiszolgálók egymással üzeneteket tudjanak cserélni. Ezek az objektumok az AD konfigurációs névterében vannak.
Ennek a két RUS szolgáltatásnak a működését sürgethetjük a System Managerből, amit a következő ábra mutat.

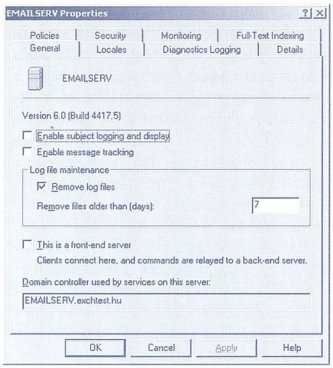


Két RUS a System Managerben

A RUS működéséről információkat kaphatunk, ha bekapcsoljuk a diagnosztikai naplózást. Ekkor az MSExchangeAL szolgáltatás 8174-es eseménye jelzi a sikeres frissítést.
Bár egy tartományban több Exchange kiszolgáló is lehet, ezek mindegyikében van RUS szolgáltatás, de csak egy aktív ezek közül.

DSAccess Service

Az Exchange 2000 működése során nagyon gyakran használja az Active Directory-t, hogy cím- és konfigurációs információkat olvasson ki. Ez sok Exchange kiszolgáló és intenzív levelezési forgalom mellett nagyon leterhelné a tartományvezérlőt és a Globális Katalógus kiszolgálóját, emellett még jelentős hálózati forgalmat is generálna. Ennek optimalizálása érdekében hozták létre a DSAccess cache-t.
Ez a szintén a System Attendant keretein belül futó szolgáltatás az Active Directory lekérdezése során visszakapott címtárobjektumokat tárolja, és újbóli lekérdezéskor már nem a tartományvezérlőtől és a Globális Katalógustól kéri be az adatokat, hanem a saját gyorsítótárából.
A szolgáltatás az inicializáláskor maximum 10 rendelkezésre álló helyi tartományi és telephelyi címtárkiszolgálót detektál, melyekhez round robin módszerrel felváltva fordul, hacsak kézzel be nem állítunk számára egyet.
A konfigurációs adatok lekérdezésére felhasznált tartományvezérlőt megnézhetjük a System Manager segítségével a kiszolgálónk „General" tulajdonságlapjának alján.



Konfigurációs DC neve a kiszolgáló tulajdonságlapján

A manuális tartományvezérlő és globális katalógus konfigurálásához a következő registry beállítások állnak rendelkezésünkre:

Helyi tartományvezérlőhöz:

```
HKEY_LOCAL_MACHINE\System\
  CurrentControlSet\Services\
  MSExchangeDSAccess\Profiles\
  Default\UserDC1 (2, ...)
IsGC = 0
HostName = Tartományvezérlő.ceg.hu
PortNumber = 0x185 (alaphelyzetben)
```

Globális katalógushoz:

```
HKEY_LOCAL_MACHINE\System\
  CurrentControlSet\Services\
  MSExchangeDSAccess\Profiles\
  Default\UserGC1 (2, ...)
IsGC = 1
HostName = GlobálisKatalógus.ceg.hu
PortNumber = 0xCC4 (alaphelyzetben)
```

Tartományvezérlő a konfigurációs adatok lekérdezéséhez:

```
HKEY_LOCAL_MACHINE\System\
  CurrentControlSet\Services\
  MSExchangeDSAccess\Instance0
ConfigDCHostName = Tartományvezérlő.ceg.hu
ConfigDCPortNumber = 0x185 (alaphelyzetben)
```

A DSAccess a megfelelő Active Directory lekérdezése után a kapott adatokat öt percig tárolja, maximum 4 MB terjedelemben. A DSAccess cache felhasználásának hatékonyságáról a System Monitor segítségével kaphatunk információt. Mind a tár méretét, mind a tárolási időtartamot tudjuk szabályozni, azonban mérlegelni kell, hogy a túl nagy tár késlelteti a frissített adatok felhasználását, a túl kicsi tár pedig nem növeli a hatékonyságot.
Ezek alapján a DSAccess gyorsítótárát a következő registry beállításokkal hangolhatjuk:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\
  Services\MSExchangeDSAccess\Instance0\
CacheTTL (REG_DWORD) – az adattárolás időtartama másodpercben
MaxEntries – maximálisan tárolt címtárbejegyzések száma
```

vagy

```
MaxMemory – a gyorsítótár maximális mérete kB-ban
```

A fentiek alapján tehát mérlegelhetjük, hogy az adattárolás időtartama mellett melyik korlátozást alkalmazzuk: a maximálisan tárolt címbejegyzések számát, vagy a gyorsítótár méretét. E két beállítás hatását úgy tudjuk összevetni, hogy a DSAccess memória felhasználása a következő képlettel becsülhető: 2.5 MB alapmemória + 3.6 kB * *(egyidejű levélterhelés)* * CacheTTL

DSProxy Service

Nemcsak az Exchange kiszolgálónak van szüksége a címtárak elérésére a levelezés során, hanem a felhasználók is lekérdezik azt a címlisták nézegetésekor. Ez nemcsak a szándékolt címlista böngészéskor valósul meg, hanem minden Outlook üzenet írásánál, hiszen az Outlook automatikusan megpróbálja beazonosítani a „To" *(Címzett)* mezőbe írt címzettet az Exchange címlistájából, még mielőtt a levelet elküldtük volna.
Outlook 2000 előtti MAPI kliensek nincsenek arra felkészítve, hogy ne az Exchange kiszolgálón magán keressék a címtárat, így ezen kliensek számára hozták létre a DSProxy szolgáltatást. Ez - a nevéből adódóan - a címtárlekérdezéseket a helyi globális katalógushoz továbbítja, majd a választ visszaadja a kérést kezdeményező kliensnek.
De nem csak MAPI klienseknél működik így a szolgáltatás. Amennyiben egy LDAP klienst az Exchange 2000-hez kapcsoljuk címtár-információk lekérdezésére, akkor is a DSProxy fogja kiszolgálni ezeket a kéréseket.
Ugyanígy, az Outlook Web Access felületet használó, felhasználók által generált http formájú címlista-lekérdezéseket az Exchange 2000 LDAP-pá konvertálja, és a DSProxy segítségével továbbítja a globális katalógushoz, majd a DSProxy-hoz érkező LDAP válaszokat html formában továbbítja az OWA klienshez.
Az Outlook 2000 kliensek alaphelyzetben másképpen veszik igénybe a DSProxy szolgáltatását. Az üzenetprofilban nem adhatunk meg külön címlista-kiszolgálót, így amikor az Outlook 2000 először indul, a profilban meghatározott Exchange 2000 kiszolgálótól megkérdezi, hogy melyik a globális katalógus kiszolgáló, akihez majd a címlista-lekérdezéseit intézheti. Ez a kiszolgáló beíródik a profilba, így innentől kezdve az Outlook 2000 mindig ezt fogja használni címlista lekérdezésre egészen addig, amíg az elérhető számára. Ha ez a kiszolgáló nem elérhető, akkor az Outlook újraindítása után új globális katalógus kiszolgálót kér a DSProxy szolgáltatástól, és ezt jegyzi be a profilba. Sajnos ez a folyamat csak a globális katalógus elérhetetlenségekor játszódik le, így a helyi globális katalógus leállása esetén, egy távoli globális katalógusra beálló kliens egészen annak elérhetetlenségéig fogja azt használni, ami nem a legoptimálisabb helyzet.

Ha ez nagyobb veszéllyel jár számunkra, mint az, hogy ezzel a folyamattal csökken az Outlook kliensek és az Exchange 2000 közti kommunikáció, ezt a hivatkozás-átadási folyamatot *(referral)* kikapcsolhatjuk a következő registry bejegyzéssel:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\
↳ Service\MSExchangeSA\Parameters
No RFR Service = 1
```

Office XP-nél még nem tudom, hogy hogyan történik ez a folyamat, érdeklődők küldjenek e-mail nekem, és ha addigra kiderítettem azt a problémát, akkor megválaszolom.

Nem nevesített szolgáltatásként futó SA funkciók

Az egyik legfontosabb további SA funkció szintén az Active Directory-hoz kapcsolódik. Hogyan is tudunk egy Windows 2000 AD-beli felhasználót levelezésbe bevonni? Az Active Directory Users and Computers eszközzel, egyszerűen elindítjuk az „Exchange Tasks" menüt a felhasználóra vonatkozóan, és ott kiválasztjuk például, hogy „Create Mailbox". Vagy kicsit bonyolultabban LDIFDE eszköz segítségével módosítjuk az adott felhasználó címtártulajdonságait, és megadunk neki egy levelező-kiszolgálót. Mindkét esetre igaz az, hogy tulajdonképpen nem közvetlenül utasítjuk az Exchange kiszolgálót arra, hogy hozza létre a levelesládát, hanem csak jelezzük a címtárban ebbéli kívánságunkat, és a System Attendant fogja ténylegesen létrehozatni a levelesládát. Ugyanígy történik a nyilvános mappák levelezésbe bevonása is.

Egy korábbi cikkemben írtam az Exchange 2000 emelt szintű biztonságáról, és ha emlékeznek, ott is szó volt a System Attendant szolgáltatásról annak vonatkozásában, hogy a felhasználóknak ez a szolgáltatás továbbítja a felhasználó-azonosító tokent, és a bizonyítványt.

Újabb fontos funkció a levelesládák *(és nyilvános mappák)* méretének figyelése és a renitensen leveleiket nem törlő felhasználók elleni fellépés. Erre szolgál a kvóták megadásának lehetősége, mellyel több szinten különböző szigorúsággal járhatunk el a kvótát túllépő felhasználókkal szemben. Első szinten kapnak egy figyelmeztető levelet, második szinten nem küldhetnek levelet, harmadik szinten nem fogadhatnak levelet. Ezeket a „büntetéseket" is a System Attendant szolgáltatás foganatosítja.

Szintén a levelesládákkal kapcsolatos a törölt elemek kétfázisú eltávolítása. Azaz amennyiben be van állítva 0 napnál nagyobb érték a „deleted items retention time", azaz a „törölt elemek visszaállíthatósága" paraméterhez, akkor a levelek törlésekor azok egy rejtett „szemetesládába" kerülnek, ahonnan az Outlook kliens megfelelő menüje segítségével visszaállíthatók. A System Attendant ebből a szemetesládából üríti véglegesen ki azokat a leveleket, melyek a visszaállíthatóságnál beállított napnál régebbiek.

Szintén System Attendanttal kapcsolatos feladat az üzenetek nyomkövetése, illetve az ezzel kapcsolatos naplóállományok kezelése. Ennek helyét szintén a registryből módosíthatjuk is.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\
↳ Service\MSExchangeSA\Parameters
LogDirectory = „Elérési út"
```

Az SA levelesládája

A System Attendant szolgáltatásnak van saját levelesládája, mely alaphelyzetben az első levéladatbázisban foglal helyet. Ennek ott van szerepe, hogy bizonyos funkciók nem működnek, ha ezt az adatbázist leállítjuk. Például az Exmerge segédprogramot sem tudjuk akkor futtatni, ha az SA levelesláda nem elérhető.

SA hibafelderítés

Ha valami problémánk adódik a System Attendant szolgáltatással, akkor érdemes bekapcsolni a kiszolgálónk System Managerbeli „Diagnostic logging" tulajdonságlapján a diagnosztikai naplózást, mellyel jóval több információ fog bekerülni az eseménynaplóba szolgáltatásunk működéséről, mint alaphelyzetben. *(Csak akkor állítható be a diagnosztikai naplózás, ha fut a System Attendant :) !)*

Kérem írják meg, hogy milyen további Exchange-dzsel, vagy SharePoint Portal Serverrel kapcsolatos témáról olvasnának szívesen!

Soós Tibor (MCSE, MCT)
soost@iqjb.hu
IQSOFT – John Bryce Oktatóközpont
www.iqjb.hu

Az Exchange Server 5.5 és 2000 vírusvédelme

Rés a pajzson – pajzs a résen?

Manapság a vállalati számítógéphálózatok vírusfenyegetettsége, sérülékenysége az internet felől igen nagy százalékban az e-mail rendszeren keresztül nyilvánul meg. A levelezőrendszerek – jelen esetben az Exchange Server – hatékony vírusvédelme komoly kihívást jelentett és még jelenteni is fog jó ideig a szoftvergyártóknak és a felhasználó vállalatoknak egyaránt. Bizonyos hiányosságok sürgős pótlást igényelnek mindkét fél részéről.

Ezen cikk szerzője nem vállalkozik a lehetetlenre, azaz minden létező Exchange vírusvédelmi megoldás pártatlan bemutatására. Azonban megpróbálok az általános technológiák és szabályok mellett konkrét termékeket is bemutatni. Mivel a Microsoft *(még)* nem gyárt saját vírusvédelmi szoftvert, a téma más gyártók termékeinek ismertetését igényli. Az említett szoftvereken kívül létezik még ezernyi más megoldás, amelyek képességeiről nem tudok *(vagy nem szeretnék)* nyilatkozni. Amit szintén tudatosan hagyok ki: az egyes AV kereső és tisztító-motorok *(engine)* hatékonyságának és felismerési képességeinek összehasonlítását. Az **[1]**-es és **[2]**-es címen található független intézetek megbízható forrást jelentenek a víruskergető motorok képességeinek minősítésére.

Az alábbiakban a témához tartozó Microsoft Tudásbázis *(Knowledge Base)* oldalakra az adott cikk azonosítójával hivatkozom Q123456 formában. Ezen azonosítók alapján a **[3]** címen lekérdezhetőek a cikkek. A vírusvédelemre a továbbiakban néha az antivírus szó AV rövidítésével hivatkozom.

Az Exchange, az Outlook, és a vírusok

A Melissa és az IloveYou megjelenése óta senkit sem érhet váratlanul az Outlook programozhatóságán alapuló féregvírusok gyors terjedése. Ezek tulajdonképpen ún. szociális vírusok, mivel a felhasználó általános érdeklődésére építve *(pl. szerelmi vallomás, vagy egy meztelen női test látványa)* próbálja rávenni a csatolt állomány megnyitására. Az utóbbi hónapokban a korábban publikusan elérhető féreggenerátor szüleményei tarolnak *(Kurnikova, Homepage, Hybris)*, komoly magyarországi informatikai cégektől is kaptam jónéhányat, nyilván „tesztelési célzattal"...

A belső és külső címlistákra továbbterjedő fertőzött csatolt állományok azonban könnyebben kivédhetőek, mint a ritkább, de az üzenetek testrészében *(message body)* jobban megbúvó html-script férgek. Míg az előbbieknél a csatolt állományok tartalomszűrése is kiegészítő megoldás lehet, a második kategóriában akár az előnézeti ablak használata is fertőzést okozhat. Ezek ellen az Outlook biztonsági beállításai és javításai hatékony védelmet nyújtanak, amelyeket az újabb verziók *(így az XP is)* már tartalmaznak, a korábbi verziókhoz pedig letölthetőek. Az Outlook E-mail Security Update *(Q263297)* az Outlook 2000 esetében egy központilag, közös mappán keresztül menedzselhető védelmi komponenst jelent.

Az Exchange kiszolgálóoldali víruskezelés eldurvulása miatt a Microsoft több saját segédeszközt is kihozott az Exchange adatbázisok tisztításához. Erre példa az Q224493-ben leírt ISSCAN is, amely megadott paraméterek alapján képes a féregvírusok üzenet és csatolt állomány hadainak kipucolására közvetlenül az Information Store-ból.

Az Exchange vírusvédelmét a rendszergazdák szemszögéből két alaptípusra oszthatjuk: az első a háttérben folyamatosan futó, az összes csatolt állományt *(és üzenetet)* módosításkor, megnyitáskor és mozgatáskor megvizsgáló ún. folyamatos vagy hozzáférési *(on-access)* komponens, míg a második az ún. igény szerinti *(on-demand)*, manuálisan vagy időzítve futtatható teljes ellenőrzés. A második kategóriába tartozó kifinomultabb szoftverek rendelkeznek inkrementális ellenőrzési opcióval, hogy a nagyméretű adatbázisokat különböző időszeletekben lehessen egyszerűbben vizsgálni. Ezt a kategorizálást termékenként lehetne finomítani, de tessék csak házi feladatnak elolvasni a termékleírásokat. :)

Az őskortól a jövőig

Ezen terület történetét az eredeti Exchange Server 5.5 helyzetével kezdem, majd rátérek a Service Pack 3 által bevezetett újdonságokra. Az Exchange 2000 kényelmetlen kezdeti helyzetén könnyítő Service Pack 1 is említésre méltó újdonságokat hoz. Az Exchange kiszolgálóoldali vírusvédelme egyre komolyabb kihívást jelent a Microsoft és az AV gyártók fejlesztőinek egyaránt, mivel az Exchange újabb verziói egyre bonyolultabbak lesznek, és az e-mail egyre központibb szerepet kap a napi, üzletileg kritikus munkafolyamatokban.

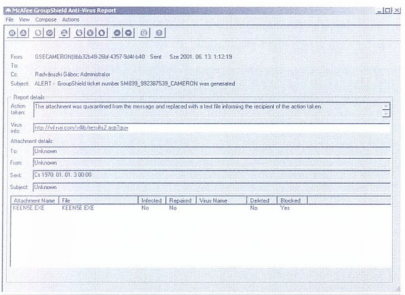
Az Exchange 4, 5.0 és 5.5 verzióinál az Exchange 5.5 Service Pack 3 megjelenéséig a vírusvédelmi szoftverek az Outlook kliensek által is használt „mezei" MAPI felületen keresztül csatlakoztak. Maga az e-mail vírusvédelem a kezdeti Exchange-s időkben még nem volt ilyen kardinális kérdés, egy 3,5"-es floppyn keringő bootvírus jellemzőbb formája volt a veszélyeknek. Azonban a MAPI-alapú fejlesztések több problémával is szembesültek, például a skálázhatósági korlátokkal. Az emberek által vezérelt kliensek *(Outlook)* MAPI kapcsolatait kellett sebességben felülmúlnia a vírusvédelmi szoftvernek, hogy még a felhasználó előtt „érjen oda" a csatolt állományhoz, ami bizonyos terhelés felett nem mindig sikerült. A széleskörű féregvírus fertőzések, amelyek párhuzamosan akár az adott Exchange-felhasználó cég összes postaládáját is érintették, végképp falhoz állították a MAPI-alapú AV szoftvereket. A skálázhatósági kihívások így azonnal biztonsági, sérülékenységi problémákká fajulhattak.

Exchange 5.5 SP3

Ettől a javítócsomagtól számítjuk az Exchange vírusvédelmi ivarérettségét, mivel a Microsoft az SP3-ba beépítette a Antivirus API (AVAPI, vagy máshol Virusscan API, VSAPI, Q263949) támogatást, mint hivatalos AV csatlakozási felületet. Ezután sok AV gyártó átírta termékeit az új csatolási felületre, és a skálázhatósági, sebesség-teljesítményi

mutatókkal próbálta rávenni a felhasználókat a frissítésre *(sokszor teljesen ingyenesen)*. Sokan azonban az SP3 követelmény miatt késlekedtek a frissítéssel. A VSAPI egyébként összecsapott, áthidaló megoldásként jött létre, amely rövidtávon skálázhatóságban és sebességben megoldást nyújtott, azonban nem tette elérhetővé se a feladó és a címzett azonosítását, se az üzenet testének és témájának *(message subject és body)* vizsgálatát, mivel csak a csatolt állományt adta át az AV szoftvernek. Az AV gyártóknak egyedi megoldást, segédprogramokat kellett írni a feladó és a címzett azonosításához, az üzenetek testében található html script vírusokat pedig kizárólag igény szerinti szkenneléskor tudták érzékelni. Az SP4 jónéhány stabilitási problémát kijavított a VSAPI implementációkon, de új funkciókat nem vezetett be. Az Exchange 2000 megjelenésekor elődjének összes vírusvédelmi hiányosságával rendelkezett. VSAPI és MAPI alapú, kombinált *(választható)* elérésű termékek jelentek meg a piacon, pl. a Symantec NAV for Exchange és a McAfee GroupShield for Exchange szoftverei. Ezek az adott környezeti problémákból eredő hiányosságokkal rendelkeznek, pl. képtelenek az Outlook Web Access-en *(OWA)*, az IMAP és POP3 klienseken keresztül kezelt csatolt állományok ellenőrzésére *(Q286638)*, valamint nem tudnak a kimenő SMTP forgalommal mit kezdeni. Súlyosabb esetben a közvetlen bejövő SMTP forgalom is átsétál rajtuk, amelyet pl. a McAfee egy külön kiegészítő SMTP szkenner programocskával hidalt át. A Symantec azt javasolja, hogy Exchange 2000 az SP1 megjelenéséig ne legyen közvetlen SMTP fogadó, hanem előbb egy dedikált SMTP kapugépen történjen meg az Exchange-független szűrés *(amely szintén mehet NAV termékkel)*. Nemcsak ezen szoftvergyártók saját fejlesztési bukdácsolásainak köszönhetőek a jelenlegi Exchange 2000-hez készült AV szoftverek korlátai, hanem az adott illesztési felület fejlesztését halogató Microsoftnak is.

A jelenlegi, SP1 nélküli implementációk *(és általában bármilyen vírusvédelem)* alapvető teszteléséhez javasolt az EICAR tesztállomány, amely a **[9]**-es címről szedhető le. Ha különböző kliensekkel *(Outlook, OWA, IMAP4)* csatolt állományként küldjük-fogadjuk, a lehetséges lyukakat hamar megtalálhatjuk.



VSAPI hiányosságok a'la McAfee (Se feladó, se címzett, se tárgy. Ez nem is csoda egy 1970-ből származó levél esetén :)

Exchange Server 2000 SP1

A MAPI funkciógazdagságát a VSAPI sebességével és skálázhatóságával kombináló VSAPI 2.0 jelenti a korrekt megoldást az égető AV problémákra. A VSAPI2 az alábbi funkciókat is támogatja majd:

- hozzáférés a feladó és a címzett paramétereihez
- a feladó és a címzett figyelmeztetése levélben
- hozzáférés az üzenet tárgyához *(subject)*, szűrés tárgy alapján
- az üzenet testének ellenőrzése *(message body)*
- mindkét irányú SMTP forgalom ellenőrzése
- OWA védelme
- az M: meghajtó kezelése
- kiválasztott postafiókok és közös mappák kizárása az ellenőrzésből

A SP1 megjelenése után közvetlenül elérhető lesz a Symantec NAV 2.5 for Exchange 2000 és a McAfee GroupShield 5.0 for Exchange 2000, amelyek az előbbiekben felsorolt teljes funkcionalitással rendelkeznek majd, ezáltal eljön majd a tejjel-mézzel folyó Exchange vírusvédelmi Kánaán.

A harmadikutas megoldások

Az eddig részletezett MAPI és VSAPI megoldásoktól radikálisan eltérő elven működik a Sybari által készített termék. Az Extensible Storage Engine *(az Information Store szíve)* nem dokumentált elérésével, cserejével egy jól működő, de a Microsoft által hivatalosan nem támogatott megoldás jött létre. A status quo-t a Microsoft Tudásbázis Q250500 cikke teremti meg, amely leírja, hogy a Microsoft Premier támogatást *(PSS)* igénylő ügyfeleknek hogyan kell ideiglenesen letiltani az ESE piszkálását. A MAPI és az AVAPI gyermekbetegségeit nélkülözi, a vírusokat az üzenetek testrészében is el tudja kapni, akár az OWA-n keresztül is. A Sybari szoftvere a víruskereső-motor szempontjából nyitott, az Exchange 5.5 és 2000 használatakor is jónéhány ismert gyártó motorjával tud együttműködni *(NAI, Sophos, Norman, CA)*. Esetében jobban oda kell figyelni azonban az Exchange hotfixek és patch-ek telepítésénél, mivel azt egyesével tesztelnie kell a Sybarinak, ami potenciális késleltetést jelenthet más biztonsági lyukak betömésénél.

Ezen „egyedi" megoldás életképességét jelzi, hogy idén márciusban a Trend Micro is megjelent egy ESE-alapú termékkel, amely viszont *(az általam hozzáférhető információk alapján)* csak az 5.5-ös verzióval kompatibilis.

Potenciális öngól*(ok)*

Ha egy adott kiszolgálógép az Exchange futtatása mellett állomány-kiszolgálóként is funkcionál *(pl., SBS)*, akkor a vírusvédelmi szoftverünk állománykiszolgálót védő komponense galibákat okozhat, ha nem jól konfiguráljuk. Ez általánosan vonatkozik minden tranzakcionális adatbázisszoftverre *(tehát pl. az SQL-re is)*, mivel a komoly méretű adatbázisok és logjaik folyamatos vírusvédelmi abajgatása a teljesítményre komoly kihatással lehet. Most azonban térjünk vissza az Exchange-hez: a vírusvédelmi szoftverünkben a háttérben állandóan futó komponensre, és az időzített szkennelésre vonatkozó beállítások között egyaránt keressük meg a „kivétel*(exclude)*" paramétert, és az Exchange összes fontos könyvtárát *(az SMTP kjü-t, az adatbázisokat és logokat)* vegyük bele, az M: meghajtóval

együtt. Ellenkező esetben a beérkező e-mail vírusos csatolt állományát az állományrendszer vírusvédelme „előzékenyen" kiszűrheti, az Exchange lelki világát meggyötörheti, vagy az M: meghajtó piszkálása esetében az adatbázist is megsértheti *(Q298924)*. Az a tény, hogy ugyanattól a gyártótól származik az Exchange és az állománykiszolgálás vírusvédelme, még nem ad automatikus felmentést ezen konfigurálási teendők alól. Állítólag lassacskán a gyártók már erre is elkezdtek odafigyelni...

A jelenleg elérhető *(a cikkben említett összes)* Exchange AV szoftver általában rendelkezik a csatolt állományok kiterjesztés alapján történő tartalomszűrésének funkciójával. Mivel a potenciálisan veszélyes csatolt állománytípusokat *(exe, vbs, bat, pif, stb...)* automatikusan karanténba helyezhetjük, máris nem kell a klisésebb vírusoktól tartanunk. Ez a grafikus kezelőfelületből paraméterezhető lehetőség sokszor erősebb védelmet biztosít, mint a napi frissítésű vírusvédelmi adatbázis és motor. A Symantec NAV esetében ezt a regisztrációs adatbázisban kell paramétereznünk, de nekik van külön tartalomszűrő szoftverük, a Mail-Gear. A GFI által gyártott Mail Essentials for Exchange a tartalomszűrés egyik nagymestere, itt a vírusvédelem mellett *(html scriptet is kiszedi az üzenet testrészéből)* felhasználónként definiálhatjuk a tiltandó kiterjesztéseket, vagy az üzenetek tárgymezőjét *(pl. ILOVEYOU)*. A Trend a Scanmail for Exchange E-Manager nevű szoftverével egészíti ki a vírusvédelmi funkciókat. A McAfee által kifejlesztett Outbreak Manager komponens *(a GroupShield része)* a tömeges, hasonló csatolt állományok viselkedése, mennyisége alapján tud eszkalációs útvonalon végigmenni, akár az Exchange leállításáig.

Ha nagy rendelkezésreállású, fürtözött Exchange rendszerhez keresünk vírusvédelmet, körültekintően vizsgáljuk meg az egyes termékeket, mert nem magától értetődő, hogy minden szoftver rendelkezik MS fürtözési támogatással... Vannak bizonyos szoftverek, melyek csak aktív-passzív párosítást tudnak lekezelni jelenleg, de az aktív-aktív üzemmód támogatása is elérhető már jónéhánynál.

Ez a harc lesz a végső...

Mint azt a szerteágazó veszélyforrások tárgyalása is mutatja, egyszerű, egyklikkelős megoldás nincsen. Javaslom hogy mindenki rendszeresen látogassa az általa használt vírusvédelmi szoftver gyártójának támogatási weboldalát, és a Microsoft Tudásbázist. Exchange verzióváltás vagy SP telepítés előtt pedig körültekintően tájékozódjon az elérhető termékek képességeiről, és ne sajnálja az időt tesztrendszer kialakítására sem.

Az Exchange 2000 bevezetése megfelelő vírusvédelem nélkül komolyan veszélyeztetheti a projektet és a teljes hálózatot is, ezért tanácsolom hogy kizárólag az SP1 és az arra épülő vírusvédelmi szoftverekkel együtt kezdjünk bele, vagy vállajuk az összetettebb megközelítést, esetleg a „harmadikutas" megoldást.

Ne feledjük, hogy a legtöbb AV gyártó előfizetéses alapon, adott időszakra biztosítja a legfrissebb programverziók használati jogát. Éljünk ezen lehetőséggel rendszeresen *(küldik CD-n, vagy letölthető jelszóvédett weboldalról)*, és ne csak az adatbázisokat és motorokat frissítsük, mivel a köztes, 0.01 verziókülönbségnyi frissítések is megkönnyíthetik az életünket. A különböző termékek teszteléséhez pedig a gyártói weboldalakon vagy a helyi képviseleteknél, forgalmazóknál elérhetőek a 30 napos próbaverziók.

A cikk végére egy pontokba összefoglalt kis tanácsgyűjtemény jutott *(Vírusvédelmi harcosok Exchange kiskátéja)*, az eddig figyelmesen olvasók átugorhatják. :)

- A konfigurálásnál a belső riasztásokat megfelelő helyre irányítsuk: pl e-mail SMS-en keresztül a rendszergazda GSM készülékére.
- Kérjünk az AV gyártóktól e-mail riasztást a súlyosabb víruskitörések esetében *(a weboldalaikon lehet kérni)*.
- Ne szégyelljünk több gyártó termékeit kombinálva használni.
- A vírusvédelemnek mindenütt jelen kell lennie (kiszolgálók, munkaállomások, web-proxy).
- Használjuk mindig a legfrissebb verziójú programot *(ne csak adatbázist és motort frissítsünk!)*.
- Az Exchange vírusvédelme a csatolt állományok és html scriptek tartalomszűrésével kombinálva hatékonyabb (exe, vbs, htm, pif, stb).
- A felhasználók oktatása ezen a területen is szükséges, de nem mindenható.
- Rendszeres (havi) tesztelés az EICAR tesztállománnyal, amely a **[9]**-es címről letölthető. Ha az Eicar-t sem tudja elkapni a védelem, akkor valószínűleg az éles vírusokat sem...
- Legyen naponta, vagy naponta 2x automatikusan frissítve a vírusvédelmi adatbázis az interneten keresztül.
- Figyeljünk oda az apróságokra: pl. a kombinált szerepkörű kiszolgálóknál a konfigurációs ütközésekre.
- Az Exchange védelme csakis átfogó IT biztonságpolitika részeként ér valamit. Ne sajnáljuk a pénzt szakember, specialista megbízására, ha házon belül nincs megfelelő emberi erőforrás.

Legeslegutoljára pedig a gyártók weboldalainak címe megtalálható a **[4]** és **[8]** közötti URL-eken.

(A szerkesztő megjegyzése: Az Event Sink-ek segítségével a W2K/E2K megjelenése óta elfogható, módosítható, eltéríthető a kiszolgálón áthaladó összes SMTP levél, még a belső levelezés is. Hogy ez a vírusírtók gyártóinak miért nem jutott eszébe?...)

[mick]

Radványszki Gábor, MCP
gabor@radvanszki.net

A cikkben szereplő URL-ek:

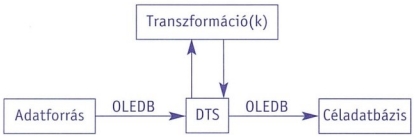
[1] http://www.virusbtn.com/
[2] http://agn-www.informatik.uni-hamburg./vtc/
[3] http://search.support.microsoft.com/kb/c.asp
[4] http://www.symantec.com http://www.sarc.com
[5] http://www.mcafeeb2b.com http:/
↳/www.avertlabs.com
[6] http://www.gfi.hu
[7] http://www.virusbuster.hu /Sybari/
[8] http://www.antivirus.com /Trend Micro/
[9] http://www.eicar.org/anti_virus_test_file.htm

Data Transformation Services *(DTS)*

A DTS funkciója

Napjaink vállalatainál még mindig rengeteg formátumban tárolódnak az üzletileg fontos adatok. Bármilyen logikus is lenne, hogy minden adatot relációs adatbáziskezelőben *(természetesen: SQL Serveren :))* tároljunk, a helyzet sok helyen egyelőre inkább a kaotikus adattárolási „modell" jegyeit viseli magán. Nemcsak hogy a clipperes programok nem haltak ki egészen, de a rendkívül flexibilis MS Access csábításának engedve egyre-másra születnek vállalati adatbázisszigetek. A szétszórva található adatok elemzése lehetetlen lenne, ha nem tudnánk azokat összegyűjteni egy központi adatraktárban. Sőt! Nemcsak gyűjtögetésről van szó. Sok esetben a „kaotikus modell" következményeinek elhárítása nélkül semmit sem ér az adatok központba cipelése – az adatokat az elemzéshez egységes formátumra kell hozni!

A Data Transformation Services pontosan a fenti feladatok elvégzésére való. Tetszőleges adatforrásból tetszőleges céladatbázisba képes adatokat mozgatni, s eközben tetszőleges adatátalakítást *(transzformációt)* végezhet.



A DTS működésének vázlata

A DTS először az SQL Server 7.0 verziójában jelent meg. Az SQL Server 2000-beli DTS bővebb szolgáltatásokat és részletesebb dokumentációt tartalmaz. Ez a cikk az SQL Server 2000 DTS lehetőségeiről szól, de a leírtak nagy része a korábbi verzióra is igaz. A DTS alapjaival egy hosszabb cikksorozatban foglalkozunk majd.

A DTS elsődleges célja az adatok kiolvasása *(egy OLE DB adatforrásból)*, átalakítása, és írása *(egy másik OLE DB adatforrásba)*. Ezt a funkciót szokás az ETL *(Extract Transform Load)* betűszóval rövidíteni. Az adatpumpa funkció mellett a DTS sok egyéb feladattípust tartalmaz, a gyakoribb feladatok listáját a következő táblázat tartalmazza:

DTS Task	Funkció
File Transfer Protocol Task	FTP letöltés
ActiveX Script Task	Script végrehajtása
Transform Data Task	Adatpumpa és transzformáció két adatforrás között
Execute Process Task	Tetszőleges alkalmazás végrehajthatása
Execute SQL Task	SQL parancsok végrehajtása
Data Driven Query Task	A bemenő adatok által vezérelt lekérdezések végrehajtása
Copy SQL Server Objects Task	SQL Server objektumok másolása, SQL Server adatbázisok között
Send Mail Task	Email küldés
Bulk Insert Task	Gyors adatbetöltés SQL Serverbe
Execute Package Task	Beágyazott DTS csomag(ok) végrehajtása

A DTS részei

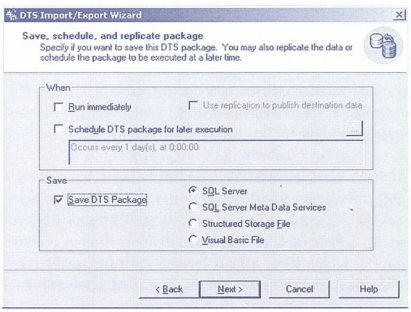
A DTS egy COM objektumokra épülő szolgáltatás az SQL Serverben. A DTS komponensek listáját a Developer Edition telepítő CD-n található redist.txt fájl tartalmazza.

Az SQL Serverrel szállított DTS alkalmazások az Import/Export varázsló, a grafikus DTS csomag szerkesztő, a DTSRun.exe parancssori végrehajtó és ennek grafikus változata, a DTSRunUI.exe. A DTS csomagok ütemezett végrehajtását az SQL Server Agent végzi.

A DTS csomag mindig annak az NT fióknak a jogosultságával fut, aki a futtatást végzi. Ha a varázslóval, vagy a grafikus szerkesztővel futtatjuk a csomagot, akkor a futtatást végző felhasználó jogai érvényesek. Ha az SQL Server Agentre bízzuk a csomag futtatását, akkor az SQL Server Agent Startup Account jogosultságaival hajtódik végre. Gyakori hibaforrás, hogy az SQL Server Agent helyi „system" fiókkal van indítva, így a távoli gépek megosztott mappáihoz nincs hozzáférési joga!

Első DTS csomagom

Valahányszor az Import/Export varázslót futtajuk, DTS csomagot hozunk létre. A varázsló utolsó lépéseinek egyikében dönthetünk, hogy az elkészített csomagot megőrizzük, vagy futás után eldobjuk. Az utóbbi az alapértelmezett működés. A következő ábra a mentési opció bekapcsolását mutatja:



A varázsló által készített DTS csomag mentése

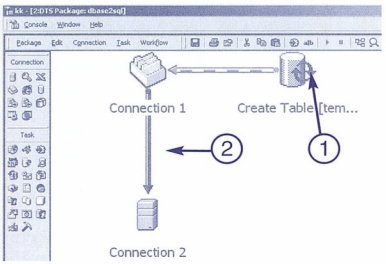
A csomagot menthetjük SQL Serverre *(az MSDB adatbázisba)*, a Meta Data Services *(repository)* alá, operációs rendszer fájlba és Visual Basic programként.
Az első két esetben a csomag a Local Packages, illetve a Meta Data Services Packages alatt jelenik meg az SQL Server Enterprise Managerben.

Az operációs rendszer fájlba mentett csomagokat a Data Transformation Services csomópontra jobb-kattintás után megjelenő menü Open Package pontjával nyithatjuk meg. A Visual Basic fájl felhasználásához a Visual Basic 6.0 szükséges.

Az SQL Server 7.0 esetén a csomagot közvetlenül nem tudjuk VB formátumban menteni. Az SQL Server 7.0 telepítő CD-n, a \devtools\samples\dts mappában található a DTSDEMO.EXE *(önkibontó)* állomány. Ez tartalmazza a ScriptPkg.vbp projeket. A ScriptPkg program segítségével tudunk SQL Serveren tárolt DTS csomagokból VB kódot generáltatni.

A DTS csomag részei

A DTS csomag lépésekből áll. A lépéseket precedenciaszabályok kapcsolják össze. A lépések hajtják végre az egyes feladatokat *(task-okat)*. A varázsló és a grafikus szerkesztő az egyes feladatokhoz automatikusan létrehozzák a lépéseket. Az alábbi csomag két lépést tartalmaz.



A DTS lépések különböző alakban „materializálódhatnak"!

Az első lépés *(Create Table, ① ikon)* egy SQL feladatot hajt végre. Ha az első lépés sikeres, következik a második *(② nyíl)*, amely a Connection1 és a Connection2 OLE DB adatforrások között végez adatmozgatást *(és esetleg adatátalakítást)*. A lépések kapcsolata lehet „Completion" „Success" és „Failure" precedencia szerint. Az utóbbi kettő *(sikeres, sikertelen)* alapján elágazásokat készíthetünk.

A DTS csomag szerkesztése, az adatpumpa

A DTS csomagok szerkesztésére, és új csomagok kialakítására szolgál a Package Editor. A varázsló által létrehozott csomagokat a Package Editorban átalakíthatjuk, vagy új csomagokat hozhatunk létre. A fenti csomagban a Connection1 és Connection2 közötti szürke vonalra duplakattintva megjeleníthetjük a Transform Data Task Properties ablakot.

A varázsló által készített egyszerű oszlopmásolás transzformációt törölhetjük, és új transzformációkat hozhatunk létre – akár oszloponként különbözőket. Új transzformáció létrehozásához először kijelöljük a forrás- és céloszlopokat, majd a New nyomógombra kattintva választhatunk a felajánlott lehetőségek közül. Íme néhány átalakítási lehetőség:

Transzformáció	Funkció
ActiveX Script	Script parancsok végrehajtása
Copy Column	Egyszerű másolás
Date Time String	Dátumformátum átalakítása
Lowercase String	Szöveg kisbetűsre konvertálása

Az ActiveX Script a legrugalmasabb átalakítási lehetőség, de – érthetően – a leglassabb is. Egyszerű átalakítások esetén célszerű a többi lehetőség közül választani.
Scriptet készíthetünk VB Script, vagy Jscript nyelven, illetve bármely script nyelven, amelynek motorját telepítettük.
A scriptszerkesztő egy identikus transzformációt generál az általunk választott nyelven *(például, VB Script esetén a következőt)*:

```
'************************************
' Visual Basic Transformation Script
'************************************
'Copy each source column to the destination column
Function Main()
  DTSDestination("ID") = DTSSource("ID")
  DTSDestination("COMPANY") = DTSSource("COMPANY")
  DTSDestination("CONTACT") = DTSSource("CONTACT")
Main = DTSTransformStat_OK
End Function
```

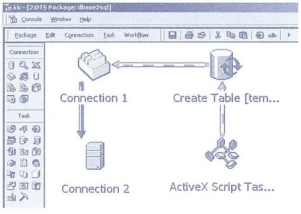
A DTSSource és a DTSDestination gyűjtemények az éppen feldolgozott sor be- és kimenő mezőit teszik elérhetővé. A script alapértelmezésben minden beolvasott sorra lefut, és egy kimenő sort hoz létre. Ezt a viselkedést befolyásolhatjuk a függvényérték módosításával. Például, ha a bejövő sor mezőit külön sorba akarjuk tenni *(mintegy „felrobbantani" a bejövő rekordot)* egy kimenő oldali táblában, amely egyetlen „col" oszlopot tartalmaz, a következő scriptet használhatjuk:

```
Function Main()
 if DTSGlobalVariables("c").value=0 or
 ↳ DTSGlobalVariables("c").value>= 3 then
    DTSGlobalVariables("c").value=1
    DTSDestination("col") = DTSSource("ID")
    Main = DTSTransformStat_SkipFetch
 elseif DTSGlobalVariables("c").value=1 then
    DTSGlobalVariables("c").value=2
    DTSDestination("col") = DTSSource("COMPANY")
    Main = DTSTransformStat_SkipFetch
 else
    DTSGlobalVariables("c").value=3
    DTSDestination("col") = DTSSource("CONTACT")
    Main = DTSTransformStat_OK
 end if
 if isnull(DTSDestination("col").value) then
 ↳ Main = Main + DTSTransformStat_SkipInsert
 End Function
```

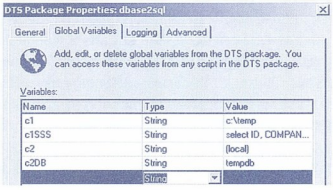
A DTSTransformStat_SkipFetch hatására a következő sor beolvasása helyett ismét az aktuális sorra fut le a transzformáció. A DTSTransformStat_SkipInsert megtiltja a keletkezett sor kiírását a fogadó táblába. *(Esetünkben a NULL kiírását akadályoztuk meg a segítségével.)*
A példascript érdekessége még a „c" globális változó használata. A globális változók lehetővé teszik az adatcserét egy csomag különböző feladatai, illetve egy külső csomag és az általa hívott csomagok között. A DTSRun futtatókörnyezet hívásakor a globális változóknak értékeket adhatunk, így a csomag végrehajtását paraméterezhetjük. A globális változókra a DTSGlobalVariables gyűjteményen keresztül hivatkozhatunk.

Paraméterezett DTS csomagok

Ha elkészült egy csomag, amely egy fájl tartalmát egy SQL Server táblába másolja, gyakori igény, hogy különböző fájlok esetén, illetve más SQL Serverekre is használható legyen. A globális változók használatával a csomag paraméterezhetővé tehető, a következőképpen. Először adjunk a csomaghoz egy ActiveX Script Taskot. Legyen az ActiveX Script Task a csomag elsőként végrehajtott lépése, az alábbi ábra szerint:



Ezután a csomag üres részére jobb-kattintva megjelenő menüből a Package Properties menüponttal megjelenítjük a DTS Package Properties ablakot. Ennek második fülében négy globális változót definiálunk:



A DTS csomagban az elsőként végrehajtódó ActiveX Script Task a globális változók alapján állítja be a bemenő fájl helyét, a bemenő lekérdezést, a kimenő táblát tartalmazó SQL Server nevét és az adatbázis nevét.
(A c1SSS tartalma a következő lekérdezés: select ID, COMPANY, CONTACT from CUSTOMER)
Az ActiveX Script Task tartalma a következő függvény:

```
Function Main()
  dim oPkg, oTask
  ' mutató a DTS csomagra
  set oPkg = DTSGlobalVariables.Parent
  ' mutató a módosítandó feladatra
  set oTask = oPkg.Tasks("Copy Data from CUSTOMER to [tempdb].[dbo].[CUSTOMER] Task")
  ' bemeneti adatforrás beállítása
  oPkg.Connections("Connection 1").DataSource =
DTSGlobalVariables("c1").Value
  ' bemeneti lekérdezés beállítása
  oTask.Properties("SourceSQLStatement").Value =
DTSGlobalVariables("c1SSS").Value
  ' kimeneti SQL Server név beállítása
  oPkg.Connections("Connection 2").DataSource =
DTSGlobalVariables("c2").Value
  ' fogadó adatbázis beállítása
  oPkg.Connections("Connection 2").Catalog =
DTSGlobalVariables("c2DB").Value
  set oPkg = Nothing
  set oTask = Nothing
  Main = DTSTaskExecResult_Success
End Function
```

A fenti kód érdekessége a DTSGlobalVariables.Parent használata. A Parent tulajdonság magára a DTS csomagra mutat. Így, miután az oPkg értéket kapott, szabadon mozoghatunk a teljes csomagban.
A csomag bemenete egy dBase fájl. Ennek helyét a „Connection 1" kapcsolat „DataSource" tulajdonsága határozza meg. A dBase fájl neve a transzformációs feladat „SourceSQLStatement" tulajdonságában megadott lekérdezés „from" záradékában található.
A kimeneti oldalon csak az SQL Server és az adatbázis nevét állítottuk be – feltételezve, hogy mindig ugyanabba a táblába fogunk beolvasni.
A csomag globális változóit futtatáskor módosíthatjuk. A következő DTSRun parancs a /A kapcsoló segítségével ad értéket a csomag globális változóinak:

```
DTSRun /S (local) /E /N dbase2sql /A"c1:8=c:\temp"
/A"c1SSS:8=select ID, COMPANY, CONTACT from
CUSTOMER" /A"c2:8=(local)" /A"c2DB:8=tempdb"
```

A globális változó neve utáni „:8" a változó típusát, esetünkben a string típust jelzi.
A DTSRun használatát részletesen ismerteti a Books Online „dtsrun utility" című fejezete.

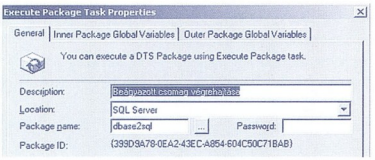
A DTS csomag szerkezetének felderítése

Honnan tudhatjuk, mit és hogyan kell módosítani egy DTS csomagban? Az egyik megoldás, hogy a csomagot Visual Basic formában elmentjük, és tanulmányozzuk a keletkező kódot. Közben időnként érdemes beleolvasni a Books Online-ba. A másik, talán egyszerűbb lehetőség a Disconnected Edit használata.
A Package Editorban a Package menüből indítható a Disconnected Edit, amely jól mutatja a csomag szerkezetét, és a csomag módosítására is használható. *(Vigyázat! A Disconnected Edit hasonló a regedt32-höz: nagyon hatékony eszköz, de helytelen használatával a csomag könnyen működésképtelenné tehető.)*

Beágyazott és mestercsomagok

Egy adatraktár feltöltéséhez általában több bemenő adatforrásból kell adatot nyernünk, és több táblába kell adatsorokat írnunk. Vannak párhuzamosan végrehajtható lépé-

sek, de vannak esetek, amikor fontos, hogy egy, vagy több feladat végrehajtódjon, mielőtt a következő lépés elindul. Készíthetünk egy nagy csomagot, ami az egész feltöltési folyamatot elvégzi, vagy több, kissebb DTS csomagot, amelyeket egy mestercsomag hajt végre.
Beágyazott DTS csomagok hívására az Execute Package Task szolgál. Az Execute Package Task fő paramétere a végrehajtandó csomag neve:



Megadhatjuk továbbá a végrehajtandó csomag globális változóinak értékét *(Inner Package Global Variables)* és exportálhatjuk, elérhetővé tehetjük a külső csomagban definiált globális változókat a belső csomag számára (Outer Package Global Variables). Az utóbbi lehetőség kényelmes paramétercserét tesz lehetővé a külső és a belső csomag között.

Ciklusszervezés

Olykor szeretnénk egy DTS csomag egyik lépését többször, ciklusban végrehajtani. Ezt a DTS futtató „becsapásával" érhetjük el. Ha egy, már végrehajtott lépés végrehajtási állapot tulajdonságát *(ExecutionStatus)* DTSStepExecStat_Waiting értékre állítjuk, akkor a DTS a lépést újra végre fogja hajtani.
A következő példában beágyazott csomagot és ciklusszervezést használunk arra, hogy egy belső csomagot egy mappa minden fájljára végrehajtsuk.
A belső csomag *(az egyszerűség érdekében)* egyetlen ActiveX Script Task-ból áll. Az script a következő:

```
Function Main()
  MsgBox DtsGlobalVariables("file").Value
  Main = DTSTaskExecResult_Success
End Function
```

A „file" globális változót nem a belső csomagban definiáljuk, hanem a külső csomagból exportáljuk. Látszik, hogy ez egy tesztcsomag – hiba lenne felhasználói akciót kezdeményezni *(üzenetet küldeni)* egy olyan DTS csomagban, amit esetleg az SQL Server Agent hajt végre, mert nem jelenne meg a felhasználói felületen az az üzenet, amelyre OK-t kell*(ene)* nyomnunk!
A külső csomag a következő feladatokat tartalmazza:

Az „Exec Inner Package" feladat a belső csomagot hívja és exportálja a „file" globális változót.
Az „Első file" feladat a „file" globális változót feltölti a c:\temp mappa egyik fájljának nevével. *(Erre a feladatra a „Scripting.FileSystemObject" és ugyanaz a technika használható, amit az alábbi, „Loop" script mutat.)*
A „Loop" script a következő:

```
Function Main()
  Dim fs, f, fl, fc, oPkg
  Set fs = CreateObject
  ("Scripting.FileSystemObject")
  Set fl = fs.GetFile
  (DtsGlobalVariables("file").Value)
   fl.Delete   ' a már feldolgozott fájl törlése
   Set f = fs.GetFolder("C:\temp")
   Set fc = f.Files
   if fc.Count>0  Then
     For Each fl in fc
      DtsGlobalVariables("file").Value =
  "C:\temp\" & fl.Name
       exit for   ' az első fájl után kilépünk
     Next
   Set oPkg = DTSGlobalVariables.Parent
   ' megismételtetjük az előző lépést,
   ' az új paraméterrel
   oPkg.Steps("DTSStep_DTSExecutePackageTask_1").
   ExecutionStatus = DTSStepExecStat_Waiting
 End If
   Set fs = Nothing
   Set f  = Nothing
   Set fl = Nothing
   Set fc = Nothing
   Set oPkg = Nothing
   Main =  DTSTaskExecResult_Success
End Function
```

A DTS tulajdonképpen egy fejlesztési platform. Lényegesen több, mint amit ebben az írásban el tudtam mondani. Többek között, nem esett szó a testreszabott DTS feladatok készítéséről, illetve a többfázisú adatpumpa használatáról. Ezek, és sok egyéb információ részletesen, példákkal illusztrálva megtalálható a Books Online-ban.

Kószó Károly
rendszermérnök
karolyk@microsoft.com

A cikkben szereplő URL-ek:
[1] http://www.swynk.com/friends/green
[2] http://msdn.microsoft.com/sqlserver

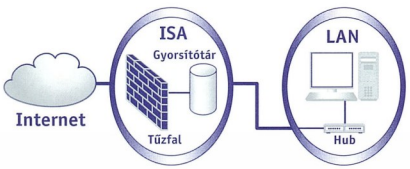
ISA Server – az alapok *(III. rész)*

Topológiák és alapszolgáltatások

E havi cikkem első részében megismertetem a Kedves Olvasót néhány gyakran használt tűzfaltopológiával, a második részben pedig már valós tesztrendszert építünk fel, melyben az ISA legalapvetőbb szolgáltatásait *(böngészés, webkiszolgáló publikálás, FTP szolgáltatás, levelezés)* fogjuk megvalósítani. Szeretnék azonban előrebocsátani néhány dolgot: a topológiarészben leírtak nem kötelezőek, mindössze néhány ötletet adnak egy tűzfalrendszer kiépítéséhez. A kiszolgálók publikálását pedig senki ne tegye éles környezetben az itt leírtak szerint! Ez úgy érzem magyarázatra szorul: a valós életben, mielőtt „bedugjuk a kábeleket", tegyünk meg mindent, hogy a lehető legbiztonságosabb rendszert állítsuk üzembe! Ez most azért marad el, mert az éles rendszerek üzembeállítása általában a hibakereséssel kezdődik, erre pedig ráérünk akkor is, amikor éles rendszert építünk, vagy már mindent tudunk, ami egy hiba elhárításához szükséges lehet.

Tűzfaltopológiák

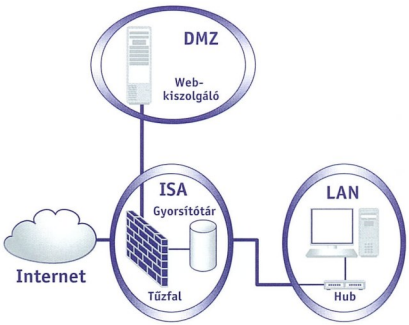
Bemelegítésként kezdjük a legegyszerűbb topológiával. Itt a tűzfal kétlábú *(vagyis két hálózati kártya van benne)*, és nem alkalmazunk DMZ-t *(erről majd később)*.



Egyszerű (router-hez hasonló) topológia

Ez a topológia leginkább kisvállalati környezetben alkalmazható, főleg olyankor, amikor nincs állandó Internet-kapcsolat, és így nincsenek a külvilágnak nyújtott szolgáltatások *(például a webkiszolgáló az Internet-szolgáltatónál van)*. Persze ez alól is lehetnek kivételek. Nagyon sokszor kényelmi, vagy éppen teljesítményokok miatt ezt a topológiát alkalmazzák, és nem alakítanak ki DMZ-t. Ez esetben a kiszolgálók a LAN-on helyezkednek el.

A következő topológia egy kicsit bonyolultabb, itt már háromlábú a tűzfal, és van DMZ is. Már sokszor említettem, lássuk hát mi is a DMZ: a DMZ a demilitarizált zóna rövidítése. Mint az nevéből is sejthető, ez egy elkülönített zóna, ami arra szolgál, hogy itt helyezzük el azokat a kiszolgálókat, melyek olyan szolgáltatásokat nyújtanak, melyeket bárki elérhet az Internet felől. Ennek legtipikusabb példája nyilvánvalóan a webszolgáltatás.



Háromlábú tűzfal DMZ-vel

Talán az ábrán nem igazán látszik, és még igencsak egyszerűnek tűnik a dolog, de azért egy ilyen rendszer beállításakor már lehetnek nehézségek. Vegyünk egy egyszerű példát: a DMZ-ben egyetlen webkiszolgáló van, amelyen egy partnereinknek szánt weblapot helyezünk el. Ezen mindig az éppen aktuális árlistánkat tekinthetik meg, ami a LAN-on elhelyezett SQL Server-ből származik.

Lássunk erre két megoldást:

1. A webkiszolgálón levő árlisták minden lekérdezéskor automatikusan frissülnek.

Mivel ez esetben webkiszolgálónk közvetlenül az SQL Server-től szerez adatokat, fontos kritérium, hogy senki ne érhesse el közvetlenül ezt a kiszolgálót, vagyis a tűzfalon egy reverse proxy-nak kell működnie. Lássuk milyen szabályokat kell létrehoznunk:

- Először is állítsuk be a webkiszolgáló eléréséhez szükséges reverse proxy-t.
- Biztosítsuk, hogy a webkiszolgáló lekérdezhessen az adatbázisból.
- És végül engedélyezzük, hogy a belső hálózatról a felhasználók arra jogosult csoportja frissíthesse a weblapokat.

Ez így igen egyszerűnek tűnik, de a későbbiekben látni fogjuk, hogy a valóságban egy-egy ilyen szabály létrehozása több lépésből áll.

2. A webkiszolgálón levő árlisták frissítését a belső hálózatról kezdeményezik *(kézi, vagy automatikus frissítés)*

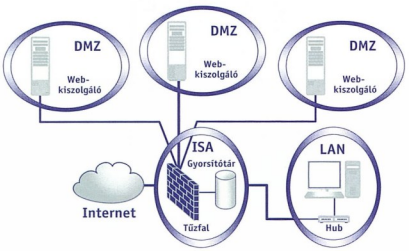
Az előző esethez képest nagyon fontos különbség az, hogy itt a DMZ-ből nem kezdeményezünk kapcsolatot a belső hálózat felé. Nyilvánvalóan ez esetben sem örömteli, ha valaki betör a webkiszolgálóra, de sokkal kisebb veszélyt jelent. Éppen ezért, hacsak teljesítményproblémák miatt másra nem kényszerülünk, használjunk itt is reverse proxy-t:

- Állítsuk be a webkiszolgáló eléréséhez szükséges reverse proxy-t, vagy az ezt helyettesítő engedélyező szabályt.

- Engedélyezzük a webkiszolgálón levő tartalom frissítéséhez szükséges forgalmakat

A két megoldás között van némi különbség. Jó, jó, ezt a vak is látja, de az első megoldás használhatóság szempontjából sokkal igényesebb, főleg akkor, ha az árlisták gyakran változnak.

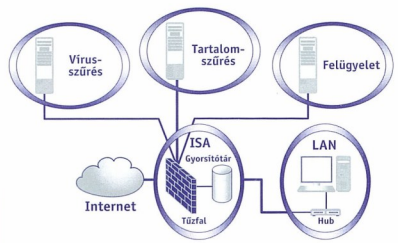
A következő topológia a paranoiásoknak való *(bár kétségtelenül vannak előnyei)*. Ez az előzőhöz teljesen hasonló, mindössze annyi a különbség, hogy itt minden egyes szolgáltatáshoz külön DMZ tartozik. *(A szemléltetés kedvéért csak plusz két DMZ-t rajzoltam, nehogy szegény tűzfal úgy nézzen ki, mint egy sündisznó :))*



Többlábú tűzfal több DMZ-vel

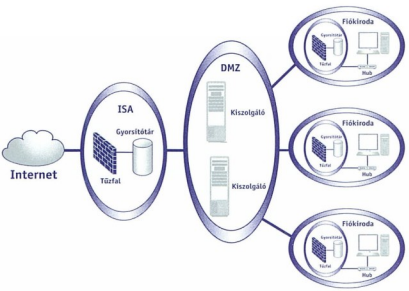
Ennek a megoldásnak nyilvánvaló hátránya, hogy bonyolultabb a tűzfalat beállítani *(gondoljunk csak például az útvonalválasztásra)*, viszont határozott előnye az, hogy ha az egyik DMZ-ben levő kiszolgálóra valakinek sikerülne betörnie, attól még nem fér hozzá ennek a kiszolgálónak a felhasználásával a többi DMZ kiszolgálóihoz.

Most induljunk ki ismét a legegyszerűbb topológiánkból *(persze az alább ismertetett megoldások használata nem zárja ki a DMZ használatát, csak szeretném, ha az ábra áttekinthető maradna)*. Létrehozhatunk olyan zónákat is, melyek olyan szolgáltatásokat biztosítanak, melyeket jó esetben senki nem vesz észre. Ilyen lehet például a vírusszűrés *(ezt azt hiszem nem kell részletezni)*, vagy a tartalomszűrés *(mielőtt olyan szoftvert szeretnénk üzembe helyezni, amely nem engedélyezi az olyan webhelyek megtekintését, amelyen egy bizonyos s-el kezdődő, és x-re végződő szó gyakran előfordul, előbb nézzük meg a naplófájlokat, nehogy a főnökség nemtetszésével találjuk szembe magunkat :))*. De létrehozhatunk olyan zónát is amelyből a tűzfalunk/tűzfalaink felügyeletét látjuk el.



Tűzfal kapcsolódó szolgáltatásokkal

Az utolsó topológia leginkább az előző részben ismertetett gyorsítótár-lánc kialakításához hasonlít. Ha a vállalatunknak több fiókirodája, vagy részlege van, melyek egymástól teljesen függetlenül működnek, lehet hogy ezeket külön tűzfal mögé akarjuk helyezni. Ha már kialakítottunk gyorsítótárláncot félig nyert ügyünk van, hiszen a szükséges eszközök megvannak, csak a megfelelő beállítások szükségesek. Ebben a topológiában a központi tűzfal mögötti terület lehet a DMZ, és ide kapcsolódhatnak a fiókirodák tűzfalai is *(Ez csak egy lehetséges megoldás. Kialakíthatjuk a rendszert úgy is, hogy létrehozunk DMZ-t/DMZ-ket a külső szolgáltatások számára, és a központi tűzfal mögötti zónában helyezzük el a fiókirodák által közösen használt kiszolgálókat)*.



Két (vagy több) tűzfal között elhelyezkedő DMZ

Most, hogy általánosságban bemutattam a legfontosabb topológiákat, térjünk rá a tesztrendszerünkre, melyet a későbbi kísérletezések során használni fogunk. Természetesen nem kötelező ragaszkodni az alábbiakban leírtakhoz, sőt! Ha valaki kísérletezgetni akar, tegye bátran, abból lehet a legtöbbet tanulni!

Az ISA tesztrendszer felépítése

ISA tesztrendszerünk központjában egy háromlábú ISA Server fog elhelyezkedni, melynek IP címeit a következők szerint osztottam ki:

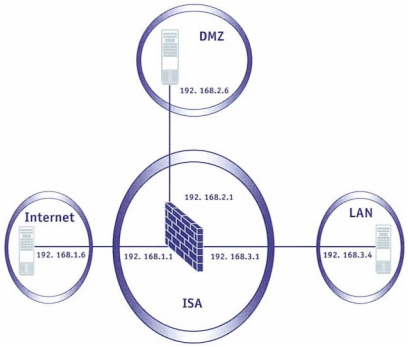
- Internet zóna: 192.168.1.1, Subnet mask: 255.255.255.248
- DMZ: 192.168.2.1, Subnet mask: 255.255.255.248
- Intranet zóna: 192.168.3.1, Subnet mask: 255.255.255.248

A szabályok kialakítása során az egyes funkciókat össze fogjuk vonni, hogy kevesebb gépre legyen szükség. Ezek szerint az Intranet zónánk egyetlen gépből fog állni:

Ezen a gépen az e cikkben leírtak megvalósításához futnia kell egy web, egy FTP és egy SMTP kiszolgálónak, az IP címe nálam 192.168.3.4 lett, alapértelemzett átjárója pedig legyen az ISA.

Az Internetet szintén egy gép fogja megszemélyesíteni, aminek az Intranet géphez teljesen hasonlónak kell lennie, de IP címe nyilvánvalóan más, legyen mondjuk 192.168.1.6, ennél a gépnél is legyen az alapértelmezett átjáró az ISA.

A DMZ-ben levő gép *(ezt most még nem fogjuk használni)*, csak a változatosság kedvéért legyen ugyanolyan, mint a másik kettő, IP címe legyen 192.168.2.6, az alapértelmezett átjárója pedig ismét csak a változatosság kedvéért, legyen az ISA.



Az ISA tesztrendszer felépítése

Most már szinte minden készen áll ahhoz, hogy elkezdjünk szabályokat létrehozni, és tesztelni azokat. Valami azonban még hiányzik! Ez a valami a DNS, de ezt most egy elegánsnak éppen nem nevezhető huszárvágással kihagyjuk a rendszerünkből *(legalábbis egyelőre)*.

Hogy fogunk tesztelni?

Mindig a lehető legegyszerűbb módon. Vagyis a megkülönböztethetőség érdekében a webkiszolgálókon az alapértelmezett nyitólap lehetőleg utaljon arra, hogy mit is látunk. A HTTP szolgáltatások kipróbálását webböngészővel, az FTP és SMTP szolgáltatásokét pedig parancssorból célszerű végezni.

Az ISA alapszolgáltatásainak beállítása

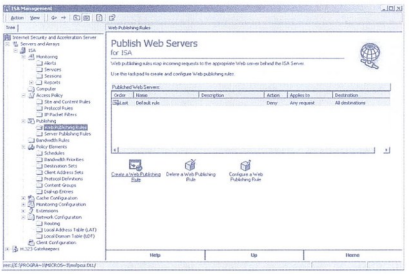
Az ISA beállításait a Windows 2000 környezetben már megszokott Microsoft Management Console segítségével végezhetjük el. Két fő fát találhatunk a jobb oldali mezőben, az egyik a „Servers and Arrays" a másik a „H.323 Gatekeepers" nevet viseli. Ez utóbbit most hagyjuk figyelmen kívül.

A „Servers and Arrays"-ben egy darab kiszolgálót láthatunk, amit én a hangzatos ISA névvel láttam el, gyakorlatilag ez alatt tudunk beállítani mindent. A főbb beállítandók a következők:

- Monitoring: itt tekinthetjük meg a felhasználók nyitott kapcsolatait, a riasztásokat, valamint a jelentéseket
- Publishing: itt tehetjük közzé kiszolgálóinkat az Interneten
- Bandwith Rules: a rendelkezésre álló sávszélesség felügyeletét biztosító szabályok
- Policy Elements: ezek képezik az ISA által kikényszerített házirendek alapját
- Cache Configuration: a gyorsítótár jellemzőinek beállítása, letöltések időzítése
- Monitoring Configuration: Jelentéskészítés és a riasztások beállításai
- Extensions: ez biztosítja az ISA bővíthetőségét, itt adhatunk hozzá külső gyártó, a Microsoft, vagy akár saját magunk által készített szűrőket
- Network Configuration: az útvonalválasztás és a belső hálózat jellemzőinek beállítása
- Client Configuration: a tűzfalügyfelekre érvényes beállítások

Webkiszolgáló közzététele

Most, hogy már alapvetően ismerjük, hogy mit hol tehetünk meg, publikáljunk is egy webkiszolgálót:

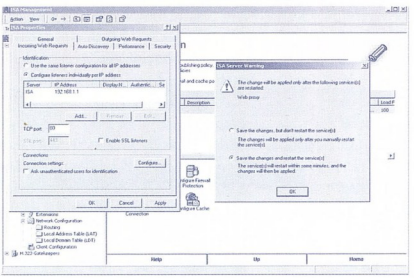


Webkiszolgáló publikálása

- Navigáljunk a Publishing\Web Publishing Rules-hoz.
- Kattintsunk a Create Web Publishing Rule-ra a jobb oldalon, a megjelenő ablakban nevezzük el a szabályt, és menjünk tovább.
- A következő képernyőn kiválaszthatjuk, hogy mely célokra *(Destination Set)* érvényesítsük ezt a szabályt, mivel még nem hoztunk létre ilyet, mondjuk azt, hogy „Any destinations".
- Ezután azt állíthatjuk be, hogy kitől is fogadunk kéréseket. Három lehetőségünk van: bárkitől, adott IP címről, vagy adott felhasználóktól és felhasználói csoportoktól. Válasszuk most az „Any request"-et.
- Ezután állíthatjuk be a szabály által végrehajtott cselekvést. Vagy elutasítjuk a kérést, vagy továbbítjuk egy belső kiszolgálónak, amelynek itt tudjuk beállítani az IP címét, és hogy milyen típusú *(HTTP, SSL, FTP)* kéréseket milyen portokra továbbítunk.

Majdnem készen is vagyunk, de ha a külső ügyféllel most próbálnánk elérni a belső kiszolgálót, akkor nem kapunk választ. Ahhoz, hogy tényleg látható legyen a publikált kiszolgálónk, be kell állítanunk az ISA külső lábán egy Listener-t *(fülelő? :))*.

- Kattintsunk jobb gombbal az ISA kiszolgálóra, és válasszuk a Properties-t.
- Menjünk az Incoming Web Requests fülre, és válasszuk ki a „Configure listeners individually per IP address"-t, és kattintsunk az Add gombra.
- A legördülő menükből válasszuk ki az ISA-t és külső IP címét, OK, aztán érvényesítsük ezt a beállítást *(Apply)*.
- A megjelenő ablakban mondjuk azt, hogy mentjük a változásokat, és újraindítjuk a szolgáltatást.



Web Listener beállítása

Próbáljuk ki bátran, a külső ügyfél most már el tudja érni a webkiszolgálót. Ha már így belejöttünk, publikáljuk gyorsan FTP és SMTP kiszolgálónkat is.

FTP és SMTP kiszolgáló közzététele

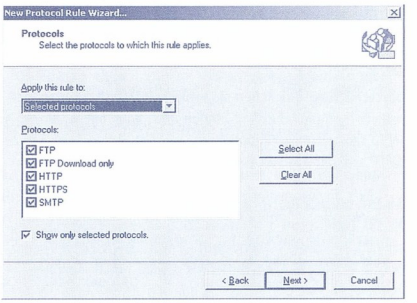
Nem véletlenül vontam össze ezt a két lépést. Az FTP és az SMTP kiszolgáló közzététele gyakorlatilag ugyanúgy zajlik, mindössze a protokoll megadásakor van eltérés.

- Válasszuk a Publishing-ből most a „Server Publishing Rules"-t.
- Nevezzük el a létrehozni kívánt szabályt.
- A következő ablakban adjuk meg a belső kiszolgáló IP címét, és az ISA külső IP címét, amelyre az adott protokoll szerinti kéréseket várni fogja *(a mi ISA Server-ünknek csak egy külső IP címe van, tehát most állítsuk be ezt)*.
- Állítsuk be, hogy milyen protokollra legyen érvényes ez a szabály, esetünkben ez értelemszerűen FTP Server, vagy SMTP Server lesz.
- A következő ablakban beállíthatjuk, hogy minden kérést elfogadunk-e, vagy csak bizonyos IP címről érkezőket, már csak egy OK-t kell nyomnunk – tulajdonképpen végeztünk is.

Szolgáltatások engedélyezése a belső felhasználóknak

Az Internet felől most már minden szolgáltatásunk elérhető, de ez saját felhasználóinkat még nem teszi túl boldoggá, mert ők még semmit nem tudnak elérni az Interneten. Hogy ezen az állapoton változtassunk, a következőket kell tennünk:

- Válasszuk az Access Policy menüből a Protocol Rules-t.
- A jobb oldalon válasszuk ki a Create a Protocol Rule for Internet Access-t.
- Nevezzük el a szabályt, és haladjunk tovább. A következő ablakban válasszuk ki azokat a protokollokat, melyeket engedélyezni akarunk a belső felhasználóknak *(ha a Show only selected protocols négyzetből kivesszük a pipát, megcsodálhatjuk, hogy milyen sok protokollt ismer az ISA Server)*.
- Állítsuk be, hogy milyen időpontokban legyen érvényes ez a szabály. Mivel még nem hoztunk létre ilyen időpontokat a Policy Elements között, mondjuk azt, hogy „Always".
- A következő ablakban beállíthatjuk, hogy kiktől is jöhetnek ezek a kérések *(bárki, adott IP címek, adott felhasználó, vagy csoport)*.
- Ismét majdnem készen vagyunk, de még át kell verekedni magunkat az ISA beépített tartalomszűrésén. Válasszuk most az Access Policy-ből a Site and Content Rules-t, majd a jobb oldalon a „Create a Site and Content Rule"-t.
- Nevezzük el a szabályt, lépjünk tovább, majd a szabály által végrehajtandó cselekvéseknél válasszuk az „Allow"-t.
- Állítsuk be, hogy mely kiszolgálókat kereshetik fel felhasználóink. Még mindig nem hoztunk létre Destiantion Set-et, tehát válasszuk az „All destinations"-t.
- A továbbiakban alkalmazzuk ugyanazokat a beállításokat, mint az előbb létrehozott Protocol Rule-nál, és hopp, már készen is vagyunk, ki lehet próbálni a szabályok működését.

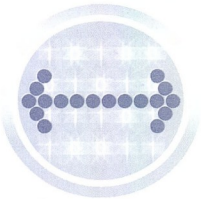


A belső felhasználóknak engedélyezett protokollok

Hát ennyi lett volna erre a hónapra. Legközelebb az ISA házirendjének beállításait fogjuk alaposabban áttekinteni *(ezzel együtt persze szűréseket fogunk végezni tartalom és időpontok szerint)*, és hogy kényelmesebben tudjuk használni tesztrendszerünket, csinálunk hozzá DNS-t is. Emellett, hogy ne legyen fölösleges az ISA-ban levő harmadik hálózati kártya, áttérünk a DMZ használatára, amit ezután már következetesen alkalmazni fogunk.

BP, MCSE

ASP suli – Komponensek *(VI. rész)*



Ezúttal az ASP programozás során hasznos, az Internet Information Server-be *(vagy az operációs rendszerbe)* eleve beépített, programozható komponensekről ejtünk néhány szót. A szöveges fájlok elérésére használt komponenst már eddig is használgattuk, de most bemutatjuk a teljes arzenált – némelyikükről sokan azt sem tudják, hogy létezik. Az adatbázis-kezeléshez használható ASP objektumokat – leszámítva a bináris fájleléréshez használt Stream objektumot – egy későbbi részben, külön tárgyaljuk majd. Az első néhány bemutatott komponens nem függ szorosan az ASP-től, ezért ezek ismerete nemcsak a web-, de a script- és Visual Basic programozók számára is hasznos lehet. A példaprogramok – mint mindig – az **[1]** címről tölthetők le.

A fájlrendszer elérése

Cikksorozatunk példáiban akarva-akaratlanul is feltűnt már a FileSystemObject **[2]** komponens. Az FSO-t a helyi és hálózati meghajtók elérésére, mappák és fájlok kezelésére *(másolás, átnevezés, stb.)*, valamit szöveges tartalmú állományok olvasására és írására használhatjuk. Az FSO objektumcsalád a következő tagokból áll:

- FileSystemObject – az objektumcsalád „őse", számos saját funkcióval
- Drive – egy *(helyi vagy hálózati)* logikai meghajtót jelképező objektum
- Drives – a számítógépen elérhető Drive objektumokat tartalmazó kollekció
- Folder, Folders – egy mappát jelképező objektum, valamint Folder objektumokat tartalmazó kollekció
- File, Files – fájlokat jelképező objektum és File objektumokat tartalmazó kollekció
- TextStream – szöveges fáljok olvasására, írására használt adatfolyam-objektum

A FileSystemObject objektum

Mindenekelőtt hozzuk létre az objektumot. A FileSystemObject komponenst a felhasználás helyén közvetlenül, vagy az <OBJECT> elem segítségével – akár a global.asa-ban – is létrehozhatjuk:

```
Set oFSO = Server.CreateObject
("Scripting.FileSystemObject")
```

```
<OBJECT RUNAT=Server SCOPE=Application ID=oFSO
PROGID="Scripting.FileSystemObject">
</OBJECT>
```

Ha az <OBJECT> formátumú definíciót nem a global.asa-ban, hanem az adott oldal tetejére szúrjuk be, ne felejtsük el a SCOPE értékét „Page"-re állítani!
Ha az oldal tetejére vagy a global.asa-ba beszúrjuk az alábbi sort:

```
<!--METADATA TYPE="TypeLib" FILE="scrrun.dll"-->
```

... akkor nem kell a típuskönyvtár elemeinek értékeit keresgélnünk, hanem közvetlenül használhatjuk az értékek neveit. A FileSystemObject objektum Drives jellemzője a Drives kollekciót adja vissza, benne az elérhető logikai meghajtókat jelképező Drive objektumokkal. Ebből kiindulva a teljes fájlrendszer bármely pontjára elérhetünk, de a GetFolder() és a GetFile() metódus segítségével közvetlenül is hozzáférhetünk egy-egy adott Folder, illetve File objektumhoz:

```
Set oFile = oFSO.GetFile("filenev.txt")
Set oFolder = oFSO.GetFolder("C:\mappa\")
```

Ha az aktuális mappát szeretnénk, írjuk ezt:

```
Set oFolder = oFSO.GetFolder(".")
```

Egy adott meghajtót pedig így érhetünk el:

```
Set oDrive = oFSO.GetDrive("C:\")
```

A GetDrive() paramétereként átadhatjuk magát a betűjelet *("C" vagy "C:")*, illetve hálózati meghajtó elérési útját is *("\\server\share")*. A GetSpecialFolder() metódus a Windowsban gyakran használt fontosabb mappák közvetlen elérésére használható *(specfld.asp)*. Paraméterként az alábbi értékeket adhatjuk át:

Név	Érték	Leírás
WindowsFolder	0	A Windows fájlok helye *(pl. C:\WINNT)*
SystemFolder	1	A Windows rendszerfájlok helye *(pl. C:\WINNT\System32)*
TempFolder	2	Az átmeneti fájlok helye *(pl. C:\WINNT\Temp)*

Ha a global.asa-ban vagy az .asp oldal tetején megadtuk a típuskönyvtárat *(a <!-- METADATA --> elem segítségével)*, akkor használhatjuk az értékek neveit, különben csak az értékeket önmagukat adhatjuk át a függvénynek. Például:

```
oFSO.GetSpecialFolder(WindowsFolder) ' = 0
oFSO.GetSpecialFolder(0) ' = WindowsFolder
```

A CopyFile(), CopyFolder(), DeleteFile(), DeleteFolder(), MoveFile(), MoveFolder() metódusok segítségével két *(az FSO objektum létrehozása után egy)* sorban másolhatunk, mozgathatunk, törölhetünk mappát vagy fájlokat, például:

```
oFSO.CopyFile("c:\boot.ini", "c:\boot.bak")
```

A DriveExists(), FolderExists(), FileExists() metódusok egy-egy meghajtó, mappa illetve fájl meglétének vizsgálatára használhatók: értékük True, ha az adott objektum létezik.
A GetTempName() metódus egy véletlenszerűen generált fájlnevet ad vissza, amit átmeneti fájlok létrehozására használhatunk – ezt értelemszerűen illik az átmeneti fájlok könyvtárában tennünk:

```
sTempName = oFSO.GetTempName()
Set oTempFld = oFSO.GetSpecialFolder(TempFolder)
Set oTempFile = oTempFld.CreateTextFile(sTempName)
```

CreateTextFile() metódussal a FileSystemObject önmaga, valamint minden Folder objektum rendelkezik. Ha a metódust az FSO-ból hívjuk, meg kell adnunk a létrehozandó fájl teljes útvonalát; míg ha egy Folder objektumból *(mint fent)*, akkor elég a fájlnevet megadni, és az az adott mappában jön majd létre. A CreateTextFile() a fájlnév mellett két opcionális paramétert vár, a második paraméter arra vonatkozik, mi történjen, ha az adott fájl már létezik. True érték esetén felülírjuk, az alapértelmezés False: ilyenkor hiba keletkezik. A harmadik paraméter a szövegformátum: True esetén a fájl Unicode, ha pedig nem adjuk meg, vagy értéke False, akkor a fájl tiszta ASCII kódolással kerül a lemezre.
Az OpenTextFile() annyiban különbözik az előzőtől, hogy ezzel a metódussal csak az FSO rendelkezik. Itt a fájlnév mellett még három paraméter található: az iomode értéke lehet ForReading *(1, olvasásra, a fájl nem írható)*, ForWriting *(2, írásra, a fájl nem olvasható)*, ForAppending *(8, a meglévő fájl végére írunk)*. A harmadik paraméter egy boolean érték, True esetén a fájlt létrehozzuk, ha még nem létezik, False esetén nem *(ez az alapértelmezés is)*. Végül, a negyedik paraméter ismét a formátumot határozza meg, TristateTrue *(-1)* esetén a fáljt Unicode, TristateFalse *(0)* esetén ASCII kódolással, TristateUseDefault *(-2)* esetén pedig a rendszer alapértelmezett beállításával nyitjuk meg. Mindkét metódus esetében csak a fájl neve a kötelező paraméter, az összes többi elhagyható.

Mind a CreateTextFile(), mind az OpenTextFile() által visszaadott objektum TextStream típusú, ami a fájlba írásra, illetve abból olvasásra használható (később bemutatjuk). Ne keverjük ezt össze a fájlokat jelképező File objektummal, ami a fájl paramétereinek lekérdezésére, módosítására, a fájlok másolására, törlésére használható.

Néhány fájlnév-feldolgozáshoz használható függvény maradt a végére. Mindegyikük tulajdonsága, hogy paraméterként egy fájlnevet várnak (a fájlnak viszont nem kell léteznie!). A namepart.asp a következő eredményt generálja:

```
C:\mappa1\mappa2\proba.txt
GetFileName() = proba.txt
GetBaseName() = proba
GetExtensionName() = txt
GetAbsolutePathName() = C:\mappa1\mappa2\proba.txt
GetParentFolderName() = C:\mappa1\mappa2
GetDriveName() = C:
```

A Drives, Folders, Files kollekciók

E kollekciók mindegyike az általa tartalmazott objektumokról lett elnevezve. Ha a Drives kollekciót lekérjük az FSO-tól, máris kipróbálhatjuk a feldolgozását *(drives.asp)*. Legegyszerűbben egy For...Each ciklusba tölthetjük be, aminek törzsében egyenként megkapjuk a Drive objektumokat; de a kollekció méretét *(.Count)* és egy-egy adott elemét is lekérhetjük *(.Item)* – a kollekciókra vonatkozó általános szabályok figyelembevételével.

A Drive objektum

Nézzük, mit találunk a drives.asp példaprogram For...Each ciklusának belsejében! Ott megtalálható az összes Drive jellemző, most csak néhány fontosabbat mutatunk be.
Mindenekelőtt itt van az .IsReady jellemző. Ha egy Drive objektum .IsReady jellemzője False értéket ad vissza, legyünk óvatosak, mert ilyenkor a legtöbb jellemző olvasási kísérlete hibát ad vissza. Ha viszont az eszköz rendelkezésre áll, sok mindent lekérdezhetünk, például:

- .DriveType – a meghajtó típusazonosítója
- .FileSystem – a használt fájlrendszer
- .TotalSize, .AvailableSpace, .FreeSpace – a meghajtó teljes, felhasználható és szabad részének mérete

A .RootFolder metódus a meghajtó gyökérkönyvtárának Folder objektumát adja vissza – és máris megérkeztünk a következő fejezethez.

A Folder objektum

A Folder objektum – nevéhez híven – egy mappát jelképez. A folder.asp példaprogram bemutatja az objektum használatát. Az .Attributes paraméter bitmezőként tartalmazza a mappa paramétereit *(ugyanez igaz lesz a File objektumra is)*, az egyes bitek értéke és jelentése az alábbi:

Leírás	Érték	Megjegyzés
ReadOnly	1	írásvédett
Hidden	2	rejtett
System	4	rendszer
Directory	16	mappa *(csak olvasható paraméter)*
Archive	32	archív
Alias	1024	parancsikon *(shortcut)* *(csak olvasható paraméter)*
Compressed	2048	tömörített *(NTFS-en)* *(csak olvasható paraméter)*

Fájl- és mappaattribútumok

Az egyes jellemzőket természetesen bitműveletek segítségével lehet kiolvasni és módosítani. Az „Archive" mező kiolvasása például így történhet:

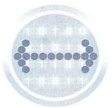
```
If oFolder.Attributes And 32 Then ...
```

Ha be szeretnénk állítani az adott értéket, akkor:

```
oFolder.Attributes = oFolder.Attributes Or 32
```

... ha pedig törölni:

```
oFolder.Attributes = oFolder.Attributes And Not 32
```



Ne feledjük el, hogy csak az írható attribútum-mezők értékeit módosíthatjuk *(bár a csak olvasható mezők írása hibát nem okoz)*. A Folder objektum metódusai segítségével másolhatjuk, törölhetjük, mozgathatjuk a mappát; a CreateTextFile() metódus pedig – már szó volt róla – szövegfájlok létrehozására való.
A Folder objektum két másik kollekciót adhat vissza: a .SubFolders jellemző az almappákat *(Folders kollekciót)*, míg a .Files jellemző a mappában található fájlok gyűjteményét *(Files kollekciót)* adja vissza.

A File objektum

A File objektum egy fájlt jelképez a fájlrendszerben. A szokásos metódusok *(.Copy(), .Move(), .Delete())* mellett egy .OpenAsTextStream() metódust is tartalmaz. A metódus által visszaadott TextStream objektum segítségével írhatjuk vagy olvashatjuk a fájl tartalmát. A File objektum is jónéhány beépített jellemzővel bír, a legegyszerűbb, ha ezeket a *file.asp* kipróbálása során ismerjük meg – és ne feledjük, a teljes FileSystemObject referencia a **[2]** címen mindig rendelkezésre áll.

A TextStream objektum

A CreateTextFile(), OpenTextFile() és az OpenAsTextStream() metódusok egy-egy TextStream objektumot adnak vissza, amely az adott – éppen létrehozott vagy megnyitott – szöveges állomány írására, olvasására használható *(textstream.asp)*. Fontos, hogy az adatfolyamban csak előre haladhatunk, visszafelé haladásra nincs mód *(ha erre lenne szükség, újra meg kell nyitni a fájlt)*. Az objektum jellemzői a következők:

- .Line, .Column – csak olvasható jellemzők, az aktuális sor, és oszloppozíciót adják vissza
- .AtEndOfLine – értéke True, ha a mutató egy sor végén áll
- .AtEndOfStream – értéke True, ha a mutató a fájl végén áll

A metódusok pedig az alábbiak:

- .Read(x) – x karaktert olvas a fájlból
- .ReadLine – egy sort beolvas a fájlból *(a sorvégjelet nem adja vissza!)*
- .ReadAll – a fájl teljes tartalmát beolvassa
- .Skip(x) – a fájl olvasásakor x karaktert lép előre *(kihagy)*
- .SkipLine – a fájl olvasásakor kihagyja a következő sort
- .Write() – a paraméterként átadott szöveget a fájlba írja
- .WriteLine() – a paraméterként átadott szöveget a fájlba írja, majd a végére egy sorvégjelet tűz
- .WriteBlankLines(x) – x üres sort ír a fájlba
- .Close() – a fájl lezárása, a munkát illik ezzel befejezni

Az ADO.Stream objektum

Bár a bevezetőben azt állítottam, hogy az ADO objektumcsaládról most nem lesz szó, egy objektum erejéig most kivételt teszek: ez pedig a Stream. A TextStream objektum ugyanis szöveges fájlok írására/olvasására úgy-ahogy alkalmas, de mit tehetünk, ha nekünk bináris állományokat kell kezelnünk? Vagy mit tehetünk ha szöveges fáljban bár, de – uram bocsá' – pozícionálni szeretnénk? Használjuk az ADO.Stream objektumot! A teljes objektumreferencia a **[3]** címen érhető el, de a fájlok kezeléséhez szükséges fontosabb jellemzőket és metódusokat mi is bemutatjuk *(stream.asp)*.
Az ADO.Stream objektumot az alábbi hívással hozhatjuk létre:

```
Set oStream = Server.CreateObject("ADODB.Stream")
```

Ezután következhet az igazi munka:

- Mindenekelőtt a Stream-et meg kell nyitnunk az .Open metódus segítségével.
- A .Type paraméter határozza meg, hogy szöveges, vagy bináris állományt kezelünk. Értéke 1, ha bináris, 2, ha szöveg – ez utóbbi az alapértelmezés is
- A .LoadFromFile(), illetve .SaveToFile() metódusok a Stream fájlból való betöltésére illetve oda mentésére használhatók
- Szöveges fájlok használatakor .charset jellemző beállítása nagyon fontos! A Stream-ből való olvasáskor, illetve abba íráskor a szöveg az itt megadott karaktertáblát használja. Az alapértelmezés ugyanis Unicode, még akkor is, ha az eredeti fájl nem Unicode típusú

Az érvényes karaktertábla-neveket a Registry-ben, a HKEY_CLASSES_ROOT\MIME\Database\Charset kulcs alatt találjuk.
Magyar szöveghez használjuk az „iso-8859-2"-t.

- A .Position jellemző mutatja, hogy éppen hol tartunk a streamben *(nincs Line illetve Column érték)* – ez viszont írható, így szabadon pozicionálhatunk – de vigyázzunk, bájtonként, nem karakterenként *(Unicode)*! A Stream a 0-s pozíciónál kezdődik.
- A .Size a Streamben található bájtok számát adja vissza. *(Unicode szöveges adat esetén ez a karakterek számának duplája!)*
- A .Read(x) illetve .ReadText(x) metódusok x bájt *(ha nem adjuk meg, teljes stream)* beolvasására használhatók. Bináris stream esetén a Read, szöveges esetén a ReadText használandó.
- A .Write() illetve .WriteText() segítségével írhatunk a Stream-be. A .WriteText második paramétere határozza meg, hogy sorvégjel kerüljön-e a kiírt szöveg végére. Ha nem adjuk meg, vagy értéke 0, akkor nem; ha azonban megadjuk *(1)*, akkor a szöveg után egy újsor karakter is kerül a szövegbe.
- Az .EOS jellemző True értéke jelzi, ha a Stream végére értünk; a SetEos() metódus pedig a Stream végét az aktuális pozícióra állítja *(a Stream többi része elveszik)*.
- A munka végén itt se felejtsük el meghívni a Stream lezárására való .Close() metódust.

A Dictionary objektum

A Dictionary objektum egy különleges tárolóobjektum *(akinek ez mond valamit: asszociatív tömb)*. Lényege, hogy benne bármilyen adatot tárolhatunk, és az egyes adatokhoz szöveges címzés segítségével *(kulccsal)* juthatunk hozzá. Például *(dictionary.asp)*:

```
Set oDic =
  Server.CreateObject("Scripting.Dictionary")
oDic.Add "kulcs1", "ertek 1"
oDic.Add "kulcs2", 123
oDic.Add "kulcs3", oDic
```

A fenti példa első sorában létrehoztunk egy Dictionary objektumot *(„Scripting.Dictionary")* majd különféle elemeket ad-

tunk hozzá, mindegyiket gondosan felcímkézve. Látható, hogy az érték lehet szöveg, számszerű érték, sőt objektum is; mi perverz módon saját magát adtuk hozzá a tömbhöz, aminek sok értelme ugyan nincs, de demonstrációs célra nagyszerű :-). Az egyes értékeket két különböző módon *(bár a két mód tulajdonképpen ugyanaz)* is kihalászhatjuk a Dictionary-ből:

```
Response.Write oDic("kulcs1")
Response.Write oDic.Item("kulcs2")
```

Azaz, használhatjuk az objektum .Item jellemzőjét, ami egy adott kulcsú értéket ad vissza, vagy írhatjuk a kulcsot közvetlenül az objektum neve után is. És hogy a rekurzió működik, mi sem jobb bizonyíték, minthogy az alábbi kifejezés eredménye 123:

```
Response.Write oDic("kulcs3")("kulcs2")
```

Néhány további jellemző illetve metódus:

- A .Count jellemző a Dictionary-ban található elemek számát adja vissza.
- A .Key jellemző egy kulcs nevének megváltoztatására használható:

```
oDic.Key("regikulcs") = "ujkulcs"
```

- Az .Exists() metódus értéke igaz, ha a paraméterként átadott kulcs már létezik.
- A .Remove() metódus egy adott kulcsú, a .RemoveAll() metódus az összes érték törlésére való.
- A .Keys() metódus a kulcsokat tartalmazó kollekciót, az .Items() metódus pedig az elemeket tartalmazó kollekciót adja vissza.

A beépített IIS komponensek

Mostantól kifejezetten az ASP programozás céljára fejlesztett objektumokról lesz szó. Az IIS ugyanis több beépített, egyszerű és bonyolultabb komponenst tartalmaz, amelyek a hétköznapi feladatok megoldásában segíthetnek. Haladjunk a legegyszerűbbtől a bonyolultabbak felé!

A MyInfo komponens

A MyInfo *("MSWC.MyInfo")* komponens eredetileg a W9x vonalhoz tervezett Personal Web Server-hez *(az IIS kistestvéréhez)* készült. A feladata az volt *(lenne)*, hogy a webkiszolgálóra globálisan jellemző adatokat tároljon *(tulajdonos, utcanév, stb.)*. Windows NT/2000-en eredetileg nem tárol semmit, de mi felhasználhatjuk, mint webszerverszerte elérhető zsákot – míg az Application objektum tartalma csak az adott ASP-alkalmazásban érhető el, a MyInfo tartalmát az egész webkiszolgáló, minden egyes webhely írhatja-olvashatja. Teljesítményokokból ASP alkalmazásonként azért csak egyet-egyet hozzunk létre, értelemszerűen a global.asa-ban:

```
<OBJECT RUNAT=Server SCOPE=Application ID=oMyInfo
PROGID="MSWC.MyInfó></OBJECT>
```

A MyInfo objektum minden bizonnyal a legegyszerűbb az IIS beépített objektumai közül, ugyanis egyetlen beépített jellemzővel vagy metódussal sem rendelkezik. Legalábbis kezdetben. Az objektum ugyanis úgy veszi fel a jellemzőket, hogy értéket adunk nekik; ha egy jellemző nem létezik, nem hibát kapunk, csak egy üres stringet válaszként *(myinfo.asp)*.

```
If oMyInfo.Kutyam = "" Then
  oMyInfo.Kutyam = "Jerry Lee"
End If
```

A beállított értékek egy XML fájlba mentődnek el időnként *(addig is a memóriában vannak)*. Ez a fájl Windows 2000 alatt a \WinNT\System32\InetSrv\Data könyvtárban található, myinfo.xml néven. Ha az oldal próbálgatása közben megnézzük, lehet, hogy még üres, de előbb-utóbb az objektumban beállított értékek a lemezre kerülnek. A fájl tartalma ilyesmi lesz:

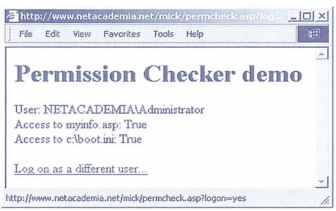
```
<XML>
<Kutyam>Jerry Lee</>
<MyNameIs>Bond, James Bond</>
</XML>
```

Az, hogy az adatok mikor kerülnek *(egyébként automatikusan)* a memóriából a lemezre, nem szabályozható, ezért viseltessünk fenntartásokkal, ha fontos adatokat szeretnénk az objektumban tárolni *(egy áramszünetet például nem biztos, hogy kibír)*.

A Permission Checker komponens

A Permission Checker *(„MSWC.PermissionChecker")* komponens segít eldönteni, hogy az oldalt letöltő felhasználónak van-e hozzáférése egy adott fájlhoz. Ezt az információt sokoldalúan felhasználhatjuk: gondoljunk csak arra, milyen diplomatikus megoldás az, ha a felhasználó számára meg sem jelenítjük azokat a linkeket, amelyekhez nincs hozzáférése.
Az objektum egyetlen metódusa, a .HasAccess() igaz *(True)* értéket ad vissza, ha a paraméterként átadott fájl létezik, és az aktuális felhasználónak van joga azt olvasni. Minden más esetben a válasz False. A paraméter lehet abszolút és relatív elérési út *(permcheck.asp)*:

```
Set oPermCheck =
  Server.CreateObject("MSWC.PermissionChecker")

oPermCheck.HasAccess( "myinfo.asp" )
oPermCheck.HasAccess( "c:\boot.ini" )
```



Az Administrator a boot.ini-hez is hozzáfér

A Counters objektum

A Counters *(„MSWC.Counters")* objektum nevéhez híven számlálók készítéséhez használható. Minden számlálót a nevével

azonosítunk. A számlálók értéke – a MyInfo leírásánál említett fenntartásokkal – lemezre kerül, mégpedig a counters.txt fájlba, a szokott helyre. A példaprogram *(counters.asp)* bemutatja az objektum felhasználását. Mindenekelőtt, a Counters objektumból is egyet érdemes létrehozni, a global.asa-ban:

```
<OBJECT RUNAT=Server SCOPE=Application ID=oCounters
PROGID="MSWC.Counters"></OBJECT>
```

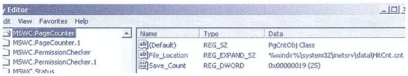
Ha ez megvan, jöhet a feldolgozás. Egy adott számláló értékét a .Get() metódus segítségével olvashatjuk ki. VBScript esetén a metódus által visszaadott értéket rögtön konvertáljuk számmá *(mondjuk a CLng() függvény segítségével)*, különben technikai okokból előfordulhat, hogy hibát kapunk. Például:

```
counterA = CLng( oCounters.Get("A") )
```

Az .Increment() metódus egy adott számláló értékét növeli eggyel, a .Remove() törli a számlálót *(a fájlból is)*, míg a .Set() metódus segítségével az adott számlálót fix értékre állíthatjuk be.

A Page Counter komponens

A PageCounter komponens *(„MSWC.PageCounter")* oldalak letöltését hivatott számlálni – természetesen az értékek lemezre is kerülnek, így nem vesznek el még váratlan leállás esetén sem. Mégpedig azért nem, mert a PageCounter objektum esetében – az előzőekkel ellentétben – beállíthatjuk, mikor írja lemezre is az adatokat. Keressük meg a registryben a HKEY_CLASSES_ROOT\MSWC.PageCounter kulcsot:



A PageCounter objektum beállításai: az adatok mentésének helye és a mentés gyakorisága a registryben beállítható

Amint az látszik, a Save_Count értéke 25. Ez azt jelenti, hogy a PageCounter 25 találat után írja lemezre az adatokat *(ez nem oldalanként 25, hanem összesen 25 találatot jelent)*. Ha ezt az értéket kisebbre *(mondjuk 1-re :-))* állítjuk, nem kell félnünk a váratlan adatvesztéstől *(azt azért várjuk meg, míg az objektum újra betöltődik és az új értéket beolvassa a registryből, például egy IIS restart hatására)*.
Maga az objektum három metódussal rendelkezik *(pagecounter.asp)*:

- .Hits() – paraméter nélkül az aktuális oldal találati számát adja vissza. Ha akarjuk, paraméterként megadhatunk más oldalt is *(virtuális ASP path segítségével: „/dir/file.asp")*.
- .Reset() – paraméter nélkül az aktuális oldal számlálóját 0-ra állítja. Paraméterként itt is megadhatjuk egy másik oldal címét.
- .PageHit() – itt nincs mese: a metódus hívásának hatására az aktuális oldal számlálója eggyel nő.

Ez az egyszerű kis számláló egy nagy hibával azért rendelkezik: nem tudja megkülönböztetni az új látogatót a notórius refresh-nyomogatótól. Ha a valósághoz kicsit közelebbi adatot szeretnénk, ki kell találnunk valamit. Süssünk egy bélyeget a látogatók homlokára *(na jó, nyomjunk sütit a szájukba, az mégiscsak humánusabb megoldás ... :-))*!

Cookie segít a számlálásban

A feladat tehát az, hogy mindenkit, aki egyszer már megbillentette a számlálót, jelöljünk meg egy cookie-val, így legközelebb már felismerjük, és nem számoljuk be az új látogatók közé. *(Azok a böngészők, amelyek nem támogatják a cookie-kat, továbbra is állandóan növelni fogják a számlálót, de bízzunk benne, hogy a többség nem idegenkedik egy kis süteménytől.)* Megjegyezhetnénk az IP címüket is, de a mai dinamikus IP címek, illetve proxyk és tűzfalak világában ennek már nem sok értelme lenne.
Küldjünk inkább egy cookie-t, ami mondjuk egy évig érvényes *(pcount.asp)*:

```
<%
  PAGEVERSION = 1
  sPageID = "counted:" &
  ↳ Request.ServerVariables("SCRIPT_NAME")
  nID = Request.Cookies( sPageID )
  If nID = "" Then nID = 0
  If nID < PAGEVERSION Then 'uj latogato
    Response.Cookies( sPageID ) = PAGEVERSION
    Response.Cookies( sPageID ).Expires =
    ↳ DateAdd("y", 1, Now() )
    oPageCounter.PageHit
  End If
%>
```

Menjünk végig soronként a fenti példakódon: A PAGEVERSION tartalmazza az oldal aktuális verzióját. Ez azért jó, mert ha az oldalon változtatunk valamit, és újra mindenkit szeretnénk számolni, elég lesz ezt az értéket megnövelni. Azután: generálunk egy nevet a cookie-nak, esetünkben ez a „counted:" szó és az adott oldal fájlneve, így nem keverjük össze, ha több oldalra is számlálót teszünk majd. Ezt a nevet az sPageID változóban tároljuk. Kiolvassuk a cookie értékét; ha nincs ilyen, értéke nyilván üres string *(nekünk jobb, ha 0)*, ha pedig van, akkor az az utoljára meglátogatott oldal verziószáma lesz. A cookie értékét az nID változóba tároljuk.
Ha a kiolvasott érték kisebb, mint az aktuális verzió, akkor egyrészt jöhet a számláló növelése *(oPageCounter.PageHit)*, másrészt pedig mehet a süti. Beállítjuk az értékét, majd a lejárati idejét mondjuk egy évre *(DateAdd("y", 1, Now()))*. És már készen is vagyunk!

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://technet.netacademia.net/feladatok/asp/6
[2] http://msdn.microsoft.com/scripting/
↳ vbscript/doc/vsobjfilesystem.htm
[3] http://msdn.microsoft.com/library/
↳ psdk/dasdk/mdao1ajx.htm

MIME – levelezés az interneten *(III. rész)*

Az elmúlt két számban megismerkedtünk a MIME üzenetek néhány fontos tulajdonságával. Tudjuk, milyen korlátokat szab az SMTP protokoll, ismerjük a MIME karakterkészletekre és tartalomtípusokra vonatkozó bővítményeit; most végre következhetnek a több részből álló üzenetek, amelyek a MIME igazi erejét adják. Az egyes RFC-k címét nem írjuk le, minden RFC ugyanúgy, az **[1]** módon leírt címmel érhető el.

A MIME Browser

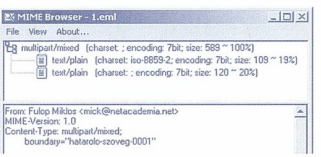
Aki Windows 2000 operációs rendszerrel rendelkezik, a **[2]** címről letöltheti a MIME Browser nevű programocskát, ami MIME formátumú levelek tartalmának vizsgálatára használható demonstrációs eszköz. A telepítés után futtassuk a programkönyvtárban található regshell.vbs scriptet, ekkor az .eml fájlok *(elmentett MIME formátumú levelek)* kiugró menüjében megjelenik a „Browse MIME Structure" menüpont, amire kattintva elindul a MIME Browser, és megkísérli betölteni az adott állományt. A cikk olvasása során ez az eszköz folyamatosan nagy segítséget nyújt majd a levélformátumok megértésében, ellenőrzésében. Annak sem kell izgulnia, aki Windows 2000 híján nem tudja használni a programot, mert az újság hasábjain ábrákkal megpróbáljuk szemléltetni a látnivalókat. Ugyaninnen, a **[2]** címről lehet letölteni a cikkben található példa-leveleket is, amelyeket bárki – még a MIME Browser hiányában is – kipróbálhat. Az .eml fájlokra kettőt kattintva az alapértelmezett levelezőprogram megjeleníti a levél tartalmát, ha pedig NotePad-del megnyitjuk őket, tanulmányozhatjuk a forráskódjukat.

Több részből álló üzenetek

A MIME üzenet több részből is állhat. Az SMTP szabvány szempontjából persze továbbra is egyetlen, csupaszöveg állományról van szó, a levél részekre bontását a szövegben található határolókarakterek *(vagy inkább „határolómondat")* segítik. Egy nagyon egyszerű, mindössze két szöveges részt tartalmazó MIME levél egyszerűsített forráskódja így néz ki *(1.eml)*:

```
From: Fulop Miklos <mick@netacademia.net>
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="hatarolo-szoveg-0001"

Ez egy magyarazo szoveg nem MIME programok
reszere. Ha ezt latja, ideje haladni a korral!

--hatarolo-szoveg-0001
Content-Type: text/plain

Ez az elso resz torzse

--hatarolo-szoveg-0001
Content-Type: text/plain

Ez a masodik resz torzse

--hatarolo-szoveg-0001--
```

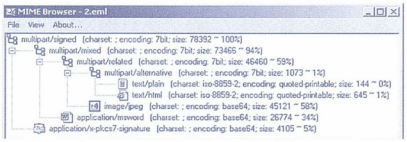
Ez az üzenet a demonstrációt leszámítva semmire sem használható, mert itt csak a MIME megértéséhez szükséges fejléceket tüntettük fel. A szokásos és elhagyhatatlan „MIME-Version" fejléc mellett feltűnik egy eddig ismeretlen tartalomtípus-főcsoport *(Content-Type)*, ez pedig a multipart. Ez a tartalomtípus jelzi a program számára, hogy az üzenet tartalma több részből áll, azt fel kell darabolni és egyenként, mint üzenetrészeket *(angolul bodypart)* kezelni. Az üzenetrészekről még később szólunk, előbb lessük meg a darabolás módját: A Content-Type fejlécnek van egy boundary paramétere *(esetünkben ennek értéke „hatarolo-szoveg-0001")*. Ahol ez a szöveg feltűnik az üzenetben *(mindig egy sorban, két mínuszjel után, lásd ①)*, ott képzeletben fogjunk egy ollót, és nyisszantsuk el a szöveget. A legelső előfordulás előtti részt *(②)* mi, MIME-ul tudók eldobjuk. Aki viszont nem ismeri a MIME-ot, annak ez fog legelőször megjelenni, ezért ez a hely ideális a régebbi levelezőprogramoknak szóló kedves üzenet elhelyezésére. A több *(nem feltétlenül két-)* részből álló üzenet utolsó részének végén található határolószöveg annyiban különbözik a többitől, hogy annak a végén is találunk két mínuszjelet *(③)*.



A levél a MIME Browserben. Jól látszik, hogy a levél két részből áll; és a részek text/plain típusúak

Az üzenetrészek

A szétszabdalt üzenet részei külön-külön egy-egy igazi, „nagy" SMTP levélhez hasonlítanak. Mindegyik üzenetrész *(bodypart)* fejrészt *(fejléceket)* és törzset tartalmaz, a két fő összetevőt pedig egy üres sor választja el. Az üzenetrész fejlécei között most persze nem a teljes levélre, csak az adott üzenetrészre vonatkozó MIME-fejlécek találhatók *(tartalomtípus, karakterkészlet, kódolás, stb.)*. Miután minden üzenetrész saját fejlécekkel rendelkezik, semmi akadálya annak, hogy az üzenet egyik része mondjuk cirill betűs szöveg, a másik pedig mondjuk egy magyar nyelvű HTML dokumentum legyen. Sőt! Az üzenetrész önmaga is lehet egy több részből álló *(multipart)* üzenetrész, saját, egyedi határolószöveggel – ilyenkor fogjuk a kisollót, és az üzenetrészt tovább daraboljuk. Akármilyen bonyolult MIME-fát építhetünk, álljon itt egy valóságból kölcsönzött, bár nem az egyszerűbbek közül való példa *(2.eml)*:



HTML formátumú, beágyazott képet és egy Word csatolt állományt tartalmazó, végül digitálisan aláírt levél felépítése

Gyakorlatlanok ne próbálják az ábrát rögtön, részletekbe menően megérteni – a migrén fájdalmas lehet, és ráadásul lassan is múlik el, – egyelőre a hierarchiát próbáljuk meg átlátni. Figyeljük meg a négy egymásba ágyazott multipart üzenetrészt, és azt, hogy ki kinek a gyermeke. A második „szinten" található multipart/mixed üzenetrész például egy Word dokumentumot és másik multipart üzenetrészt tartalmaz, ami egy újabb multipart-ból és egy JPEG képből áll – és ez még csak a hierarchia fele!

Üzenetrész-fejlécek

A multipart tartalomtípus is „csak" egy főcsoport, és ennek a főcsoportnak is több alcsoportja létezik. Mielőtt azonban a multipart-alcsoportokra áttérnénk, bemutatjuk a leggyakoribb, multipart-alcsoporttól függetlenül használt fejléceket.

A Content-Type üzenetrész-fejléc

Ez talán a legfontosabb információ: az üzenetrész által hordozott adat típusát határozza meg *(ami ugye lehet valamelyik hagyományos, az előző számban már bemutatott tartalomtípus, vagy akár egy másik multipart üzenetrész is)*. Szöveges tartalomtípus esetén a legtöbb esetben ez a fejléc rejti az üzenetrészben használt kódtábla azonosítóját *(charset)*:

```
Content-Type: text/plain;
        charset="iso-8859-2"
```

A name paraméterben pedig gyakran hordozza az üzenetrész által tartalmazott fájl eredeti nevét, például:

```
Content-Type: application/msword;
        name="doc1.doc"
```

Végül ne feledkezzünk el arról, hogy a multipart tartalomtípus esetén itt kell megadnunk a darabolás helyét jelző szöveget *(boundary)*:

```
Content-Type: multipart/mixed;
        boundary="hatarolo-szoveg-0001"
```

A Content-Type fejlécnek e háromnál sokkal többféle paramétere lehet, ezeket a paramétereket mindig maga a tartalomtípus határozza meg. A különböző multipart tartalomtípusok paramétereiről a megfelelő helyen mi is említést teszünk, egyéb esetben az RFC-k böngészése segíthet.

A Content-Transfer-Encoding üzenetrész-fejléc

Csakúgy, mint a teljes üzenetre vonatkozóan *(lásd a teljes leírást az előző számban)*, üzenetrész-fejlécként használva is az üzenetrész-tartalom 7 bitre kódolásához használt módszert jelzi. Üzenetrészenként használva megoldható, hogy az üzenet szöveges részeit a Quoted-Printable, míg a bináris részeket *(mondjuk egy csatolt word dokumentumot)* az ilyen tartalomnál hatékonyabb Base64 kódolással vigyük át. Fut még a 7bit kódolás is, ami a nemkódolást jelenti *(ASCII szöveg esetén)*. Példák:

```
Content-Transfer-Encoding: quoted-printable
Content-Transfer-Encoding: base64
Content-Transfer-Encoding: 7bit
```

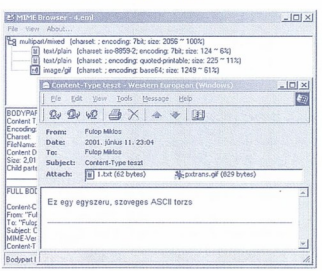
A Content-Disposition üzenetrész-fejléc

No, ez a fejléc már teljesen új. Az üzenetrészre aggatott Content-Disposition fejléc jelzi, hogy az egy csatolt állomány *(ez esetben értéke attachment)*, vagy a levélben valahol felhasznált komponens *(ilyenkor inline)*. Az inline típusú üzenetrészt a levelezőprogramok elvileg nem jelenítik meg, mint csatolmányt. Próbáljuk csak ki, nyissuk meg a 3.eml példát! A levelezőprogram a három üzenetrészből az elsőt, a levél törzsét megjeleníti; a másodikat csatolt fájlként jelzi *(kattintással megnyithatjuk, ha akarjuk)*, míg a harmadik ismét, felhasználói közbeavatkozás nélkül látható *(hiszen „része" az üzenetnek)*. Ha a Content-Disposition értéke attachment, akkor a fájl későbbi kezelésére vonatkozó információs paramétereket is tartalmazhat: mindenekelőtt a fájl nevét *(filename)*, esetleg a méretét, vagy a létrehozásának dátumát, például:

```
Content-Disposition: attachment;
        filename="1.txt"
```

Ebből tudja a levelezőprogram, mi volt a csatolt fájl eredeti neve. Ha ez a paraméter hiányzik, a program kénytelen a hasára csapni; ilyenkor kapunk ilyen fájlneveket, hogy: att00891.txt vagy hogy att00915.gif. Hogy a kiterjesztést honnan tudja mégis? Ennek megfejtése házi feladat!

Nos körülbelül ennyit az általános üzenetrész-fejlécekről. A továbbiakban a különféle multipart-tartalomtípusokról, és az általuk használt, speciális üzenetrész-fejlécekről értekezünk. Mielőtt beleugranánk a mélyvízbe, ellenőrizzük az általános üzenetrész-fejlécekről összegyűjtött tudásunkat a 4.eml példa segítségével!



Egyszerű, csatolt fájlokat tartalmazó levél – figyeljük meg az üzenetrészeknél használt különféle kódolási módszereket (encoding)

A multipart-alcsoportok

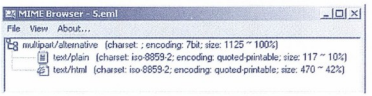
Futólag már említettük, hogy a multipart tartalomtípus-főcsoport különféle alcsoportokat tartalmaz. Az rendben van, hogy az üzenet több részből áll, de van-e a részeknek valami közük egymáshoz? Az alcsoport szerepe, hogy megjelölje az általa tartalmazott üzenetrészek egymáshoz való viszonyát. Mindegyik tartalomtípusnak különleges szerepe van, és a legtöbb saját üzenetrész-fejléceket is használ.

A multipart/mixed tartalomtípus

A legáltalánosabb, legegyszerűbb és talán leggyakoribb típus *(RFC 1521)*. A részeknek nincs szorosabb közük egymáshoz, hacsak nem az, hogy egy üzenetbe kerültek. Tipikus példa az egyszerű, csatolt állományokat tartalmazó üzenet *(mint az eddigi példák többsége, például a legutolsó 4.eml is)*.

A multipart/alternative tartalomtípus

Ez egy nagyon érdekes típus *(RFC 1521)*; szerepe a HTML levelek megjelenésével vált egyre fontosabbá. Az alternatív üzenetrészek ugyanazt tartalmazzák, különböző formátumban; közülük elég azt megjelenítenünk, ami a szívünknek kedves. Lássuk csak egy egyszerű, tisztán szöveget tartalmazó, de HTML levél felépítését *(5.eml)*:



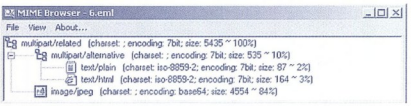
HTML levél, text alternatívával

Egy jólnevelt HTML levél egy text/plain és egy text/html részt tartalmaz, ahol a text/plain üzenetrész a HTML rész szöveges kivonatát tartalmazza. Mi dönthetünk: ha nem akarjuk a HTML-t megjeleníteni, ott a szöveges verzió. Ez persze azt jelenti, hogy a levél tartalma redundáns, viszont levelezőprogram-barát. Látható, hogy az üzenet teljes terjedelmének 42%-át a HTML üzenetrész teszi ki, a plusz szöveges rész csak további 10% *(a 100-ból fennmaradó részt a levél SMTP fejlécei képezik; ez az arány persze levelenként eltérő)*. A többlet terhelés tehát jelen van, de általában nem számottevő.

Fontos megjegyezni, hogy az alternatív tartalomtípus sem csak két verziót tartalmazhat; előfordulhat akár egy text-rtf-word6-word2000 csomag is. Az RFC 1521 mindössze annyit kér, hogy a legegyszerűbb formátum *(esetünkben a sima szöveg)* legyen a levélben legelöl, majd növekvő bonyolultsági sorrendben követhetik a további verziók. Így sokkal könnyebb a feldolgozás azok számára, akik csak valamely „gyengébb" formátumot ismerik.

A multipart/related tartalomtípus

Egymáshoz kapcsolódó üzenetrészek *(általában az egyikben hivatkozás történik a másikra)* azonosítója *(RFC 2112)*. Egyszerűbb megérteni, ha magunk elé képzelünk egy HTML üzenetet, ami egy képet tartalmaz *(tehát nem csatolt állományként, hanem beillesztve)*. A kép is az üzenetben utazik, a HTML kódban pedig speciális hivatkozás szerepel a helyén. Ha megnyitunk egy ilyen levelet, a kép az üzenet belsejében jelenik meg, és nem érjük el, mint csatolt állományt. Természetesen maga a HTML rész sem mint tiszta HTML, hanem mint komplex multipart/alternative rész utazik *(lásd az előző bekezdést)*; egy beillesztett képet tartalmazó HTML levélben tehát már felismerhetjük az első hierarchikus jellemzőket:



HTML levél, melyben kép is van

Egy kérdés nyitott maradt: arról volt szó, hogy a HTML tartalom hivatkozik a levélbe ágyazott képre. Pontosan hogyan is történik ez? Lássuk a 6.eml forrásának erre vonatkozó részeit *(mondjuk a HTML üzenetrészt teljes egészében ④, és a kép üzenetrészének elejét ⑤)*:

```
...
--hatarolo_szoveg_0001
Content-Type: text/html
Content-Transfer-Encoding: 7bit

<HTML><HEAD></HEAD>
<BODY>
<IMG src="cid:content_id_0001">
</BODY></HTML>

--hatarolo_szoveg_0001--

--hatarolo_szoveg_0000
Content-Type: image/jpeg;
        name="MSGIRL2.jpg"
Content-Transfer-Encoding: base64
Content-ID: <content_id_0001>

/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAYEBQYF...
FhYaHSUfGhsjHBYWICwgIyYnKSopGR8tMC0oMCUo...
...
```

④ ⑦ ⑤ ⑥

A határoló szövegeket próbáljuk meg magunk értelmezni, de ehhez a teljes kódot látni kell! *(Annyit segítek, hogy a _0001 végű a multipart/alternative, a _0000 végű pedig a multipart/related határolószövege.)* A lényeg most az üzenetbe ágyazott elemek „címzése". Figyeljük meg, hogy a képet tartalmazó üzenetrész tartalmaz egy Content-ID fejlécet ⑥:

```
Content-ID: <content_id_0001>
```

A fejléc értéke az adott üzenetrész azonosítója *(a kacsacsőrök nem tartoznak az azonosítóhoz)*.

A HTML kódban pedig a kép forrásaként nem internetes URL *(például http://...)*, hanem egy speciális címzés szerepel *(egyébként ez is egyfajta URL)* ⑦:

```
<IMG src="cid:content_id_0001">
```

A megjelenítéskor a levelezőprogram a cid: bevezetőből tudja, hogy a hivatkozott elemet az üzeneten belül, a megadott Content-ID alapján kell megkeresnie.

A multipart/related tartalomtípusnak van egy kötelező type paramétere, ami a „gyökérként" feldolgozandó üze-

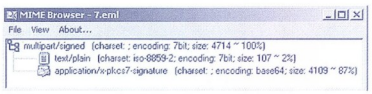
netrész tartalomtípusát tartalmazza *(annak az üzenetrésznek, amelyre nem hivatkoznak, hanem ő hivatkozik)*; esetünkben ez a multipart/alternative:

```
Content-Type: multipart/related;
        type="multipart/alternative";
        boundary="hatarolo_szoveg_0000"
```

A tartalomtípus többi paramétere opcionális, akit érdekel, olvassa el a témáról szóló RFC 2112-t.

A multipart/signed és multipart/encrypted tartalomtípusok

Digitálisan aláírt *(signed)* / titkosított *(encrypted)* üzenetek *(RFC 1847)*. Kezdjük a végéről: a multipart/encrypted, azaz titkosított többrészes üzenet, amely mindig pontosan két részből áll. Az első a titkosításhoz szükséges kontrollinformáció, a második pedig maga a titkosított adatfolyam, amely application/octet-stream *(8 bites adatfolyam)* tartalomtípust hordoz. A manapság használatos titkosítások esetén általában a multipart/encrypted tartalomtípus nem szerepel, az S/MIME levelezőprogramok elintézik a dolgot egyetlen önhordó *(még csak nem is multipart)* titkosított levéllel *(amelynek tartalomtípusa nemes egyszerűséggel application/x-pkcs7-mime)*.
Viszont az aláírt levelek! A 7.eml egy egyszerű digitálisan aláírt szöveges üzenet. A multipart/signed is mindig pontosan két üzenetrészt tartalmaz.



Digitálisan aláírt szöveges üzenet

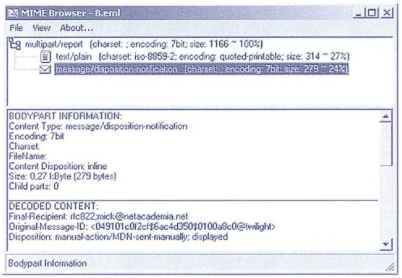
Ebből az első a digitálisan aláírt, „védett" üzenetrész *(bármi lehet, egyszerű szövegtől az összetett multipart üzenetrészekig)*; a második pedig az aláírás maga, tartalomtípusa az aláírás algoritmusától függ. Ha belenézünk a forráskódjába *(most már ideje lenne letölteni azt a MIME Browsert, nem?)*, a fő tartalomtípus fejlécénél ezt látjuk:

```
Content-Type: multipart/signed;
        protocol="application/x-pkcs7-signature";
        boundary="hatarolo-szoveg-0000";
        micalg=SHA1
```

Jól látható a tartalomtípus három kötelező paramétere. Az egyértelmű határolószöveg *(boundary)* mellett meg kell adni a digitális aláírás protokollját *(protocol, ez lesz a második üzenetrész tartalomtípusa is)*, valamint az ellenőrzösszeg képzéséhez használt algoritmus *(Message Integrity Check Algorythm, micalg)* nevét. A fejléc tanulsága szerint a példában látható digitális aláírás az SHA1 algoritmus segítségével készült. Ha az első, védett üzenetrészben bármit *(akár egy üzenetrész-fejlécet is)* megváltoztatunk, a digitális aláírás azonnal „elromlik" és kiderül a turpisság.

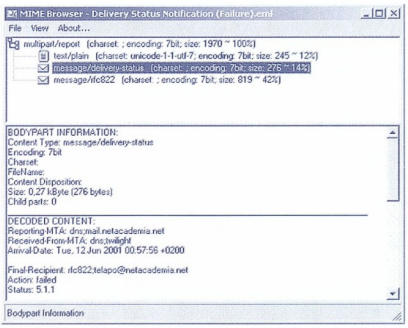
A multipart/report tartalomtípus

Bizonyára mindannyian kaptunk már valamelyik levelezőkiszolgálótól a levelünk sikertelen vagy akár sikeres kézbesítésére vonatkozó válaszlevelet, esetleg visszajelzést arról, hogy a címzett elolvasta a levelet. Azt már kevesebben tudják, hogy ezeknek a leveleknek speciális formátumuk van, elősegítve ezzel az ilyen üzenetek automatikus feldolgozását.
Válasszuk ketté a multipart/report *(RFC 1892)* tartalomtípust! Nézzük meg először, hogyan néz ki egy elolvasást visszaigazoló üzenet *(read receipt, 8.eml)*:



Read receipt

A multipart/report első tagja egy egyszerű szöveges üzenet arról, hogy elolvastam a magamnak küldött levelet. A második tag pedig egy message/disposition-notification tartalomtípusú üzenetrész *(RFC 2298)*, aminek tartalma elárulja az eredeti üzenet azonosítóját *(a levelezőprogramunk tudná követni, hogy melyik levélről érkezett visszajelzés!)*, hogy végülis ki kapta meg az üzenetet *(például ha átirányítás történt)*, illetve a visszajelzés elküldésének körülményeit *(nem automatikusan, hanem az én megkérdezésemmel történt)*.
Lássunk egy kézbesítési üzenetet *(Delivery Report)*, amelyben a levelezőkiszolgálónk értesít róla, hogy a Télapónak egyelőre nem a NetAcademia-hoz kell címezni a leveleket:



A Télapó nem nálunk dolgozik...

Az első rész itt is egy szöveges üzenet a felhasználónak. Az üzenetben szerepel még a szóban forgó levél egy-az-egyben *(message/rfc822, lásd RFC 1521)*, valamint maga a kézbesítésről szóló üzenet *(message/delivery-status, RFC 1894)*.

Most lapozzon vissza a 30. oldalra és tekintse meg még egyszer azt a bizonyos HTML levelet, amely beágyazott képet és egy csatolt Word dokumentumot tartalmaz, és a végén még jól alá is írtuk. Ugye, hogy mégsem annyira bonyolult?

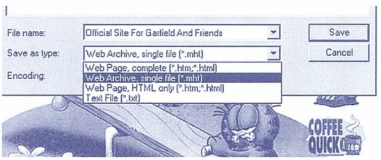
Egy csendes megjegyzés: a MIME annyira jól ki van találva, hogy az is képes a leveleket olvasni, aki kénytelen csupán karakteres (linux/unix) környezetben dolgozni. Még a fenti, igencsak összetett példa is tartalmaz tisztán szöveges összetevőt! Mindössze egy rendes, MIME-kompatibilis levelezőprogram kellene hozzá...

Az MHTML dokumentum

Egy érdekességet szeretnék mutatni a MIME-ról szóló cikksorozat végén, aminek érdekes módon a levelezéshez semmi, a MIME-hoz viszont annál több köze van. A most leírtakat az Internet Explorer 5.0 vagy újabb változatának felhasználói már ismerhetik *(illetve ha még nem ismernék, akkor kipróbálhatják)*, aki nem rendelkezik ezzel a verzióval, annak itt az ideje frissíteni. Kezdjük az alapproblémával: egy HTML oldal tartalmát szeretnénk elmenteni, későbbi olvasgatásra. A klasszikus módszer valami ilyesmit eredményez:

Ha egy weblapot elmentünk, a HTML fájl mellett a „sallangok" egy külön könyvtárba kerülnek

Van eset *(a Garfield például pont ilyen :-))*, hogy nem gond, sőt előny, ha egy weblapon található képek, grafikák külön-külön a lemezre íródnak. Máskor azonban kifejezetten macera a fájl és a hozzá tartozó könyvtár együttes másolgatása *(próbáljuk ki, és hozzunk létre egy könyvtárat és egy HTML fájlt, mint a fenti példában; azután töröljük ki a HTML fájlt, és csodák csodájára a Windows a megkérdezésünk nélkül törli a (szerinte) hozzá tartozó könyvtárat is .. khm..)*. Egyszóval ezzel sokszor csak a baj van.
Nem lehetne összecsapni az egészet egyetlen fájlba? *(Na jó, nem a tömörítésre gondolok...)*. Dehogynem. Az Internet Explorer 5.0-tól kezdve a Save As... ablakban egy eddig ismeretlen, új formátumot is kiválaszhatunk:

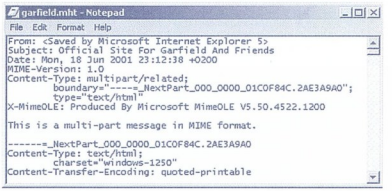


IE5-től kezdve Garfield-ot .mht-ba is menthetjük

„Web Archive, single file" *(webarchívum, egyetlen fájl)* – mondja a magyarázat az .mht kiterjesztés mellett. Nosza, próbáljuk ki! A mentés végén valóban egy fájl keletkezik *(néhányat a* **[2]** *címről is letölthet a lelkes Olvasó)*. Hurrá! Semmi sallang!

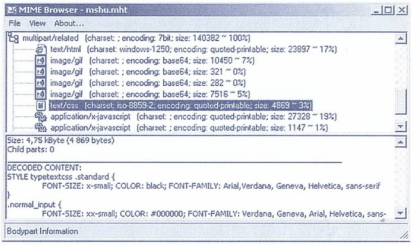
Na de mi köze ennek a MIME-hoz?!

Rögtön meglátjuk, ha egy .mht fájlt a jó öreg MIME Browser-be vontatunk *(persze akkor is kiderül a turpisság, ha egyszerűen megnyitjuk, mondjuk NotePad-del)*. Az .mht fájl *(MHTML dokumentum)* nem más, mint egy MIME formátumú HTML „levél", beágyazott objektumokkal.



Az .mht fájl valójában egy HTML levél, az M betű a MIME-ot jelképezi: MHTML Document

Még feladója, feladási dátuma és témája is van! És hogy mi van belül? Rögtön meglátjuk, ha megnyitunk egy ilyen fájlt a MIME Browserrel:



A Microsoft Magyarország elmentett weblapja a MIME Browserben

Az ábrán a Microsoft Magyarország .mht-ba mentett főlapjának kicsit kozmetikázott képe látható *(az itt láthatónál sokkal több kép van a fájlban, de ez ne zavarjon meg senkit)*. Az üzenet típusa értelemszerűen multipart/related *(aki nem érti miért pont ez, lapozzon vissza)*; az első elem természetesen maga a HTML kód, majd azt követik szépen a hozzá kapcsolódó üzenetrészek: képek, stíluslapok, javascript-darabkák.

Fülöp Miklós
mick@netacademia.net

A cikkben található URL-ek:
[1] RFC xxxx: http://www.ietf.org/rfc/rfcxxxx.txt
[2] http://technet.netacademia.net/download/MIME

Transact SQL *(VIII. rész)*

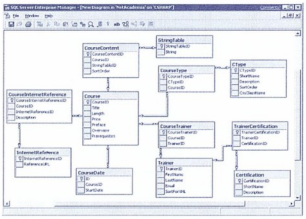
Bevezetés

Fejlesztői szemmel nézve az SQL Server 2000 egyik legnagyobb újdonsága az XML támogatás megjelenése volt. A termék megjelenése óta eltelt több mint egy év, és természetesse nincs megállás, készül a következő verzió, aminek Yukon a kódneve, és a pletykák szerint a felhasználása jelentősen túlmutat majd a jelenlegi verzióén. De az SQL 2000-et sem hagyták magára, újabb és újabb *(ingyenes)* kiegészítések készülnek hozzá, amelyek segítségével újabb képességekkel ruházható fel a szerverünk. Ezeket, és a termékben már eleve meglevő tudást járjuk körbe ebben a cikkben.

Egy hétköznapi probléma

Informatikai sznob divat vagy valóban hasznos segítség az XML támogatás az SQL Serverben? Aki volt a május végi magyar .NET fejlesztői konferencián, az láthatta, hogy a transzparenseken nagyobbak voltak az XML feliratok, mint a .NET-et hirdető igék. Az XML fenekestül felforgatja, átalakítja azt a képet, ahogyan az együttműködő rendszereket látjuk és programozzuk, és az elosztott rendszereket tápláló egyik legfőbb bástyánk, az SQL Server sem térhetett ki a változás elől. Egy egyszerű és mindennapi példán keresztül megvilágítom, hogy mennyivel egyszerűbbé válik az életünk, ha az adatainkat XML folyamként is kinyerhetjük az SQL Serverből.

Legyen a feladatunk egy webalkalmazás megírása, amely SQL Serverből származó információkat jelenít meg egy ASP alapú weblapon. A szemléletesség kedvéért tegyük konkrétá a példát. A megjelenítendő információ egy-egy fejlesztői tanfolyam leírása. Egy tanfolyamhoz nagyon sokféle információ kapcsolódik, – a tanfolyam leírásán keresztül a képzést tartó oktatók neve, képzettsége, a helyszín, időpontok, szabad helyek száma, satöbbi. Mivel az adatbázisban nem csak egy tanfolyamot tárolunk, ezért az adatbázis sok táblából fog állni, rendesen normalizálva van, így minden információt csak egyszer kell letárolni.



Tanfolyamokat leíró adatbázis-séma

Az ábrán láthatunk egy a problémára viszonylag részletes leírást nyújtó adatbázist. Látható, hogy a központi tábla a Course, és az összes többiben található információ erre a táblára illeszkedve épül fel. A feladat tehát az, hogy egy adott kurzussal kapcsolatos minden információt szedjük össze a táblákból, és jelenítsük meg egy weblapon.

Mit jelent az, hogy összeszedni? Milyen művelettel nyerhetőek ki egy adott tanfolyamhoz tartozó járulékos információk? Természetesen JOIN-nal, remélem, nem leptem meg senkit. Azaz összekapcsolom a táblákat a diagramban is jelölt kapcsolatok mentén, és máris kész egy olyan eredményhalmaz, ami könnyen megjeleníthető.

Biztos ez? Sajnos nem. Nem kell hozzá nagy képzelőerő, hogy kitaláljuk, az így összekapcsolt tábla pókhálóból nagyon sok sor fog legenerálódni amiatt, hogy az eredetileg hierarchikus információt síkba kiterítve tudjuk csak visszaadni. Az eredményhalmaz kétdimenziós, tábla szerkezetű, ha tetszik, ha nem.

Például minden tanfolyam annyiszor fog megjelenni a listában, ahány alkalommal megrendezzük, szorozva ahány oktató tartozik hozzá, szorozva ..., azaz a szimpla JOIN-ok így, ebben a formában nem megfelelőek a feladat megoldására. Azt nem mondom, hogy az így nyert hatalmas eredményhalmazból nem lehet kiszedni a számunkra értékes infókat például egy intelligens script segítségével, de a kódspagetti *(programnak nem nevezném)* áttekinthetetlen, karbantarthatatlan lesz, nem beszélve az indokolatlanul nagy eredményhalmaz okozta felesleges erőforráspazarlásról.

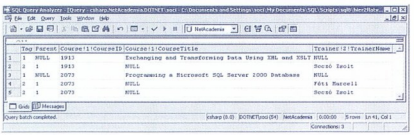
Hierarchikus kétdimenziós tábla?

Valahogyan úgy kellene felépíteni az eredményhalmazt, hogy az tükrözze az eredeti információ belső szerkezetét. Menni fog? Kicsit körülményesen, de igen. A vezérfonalunk az lesz, hogy úgy építünk fel egy táblát, hogy a sorok a hierarchia bármely szintjén levő információt meg tudják jeleníteni, és az első két oszlopot használjuk fel a hierarchia leírására. Például ragadjuk ki a Course, CourseTrainer, Trainer táblákat. A Course egy tanfolyam főbb adatait tartalmazza, a Trainer az oktatókat tároló tábla, míg a kurzusok és az oktatók közötti viszony leírására szolgál a CourseTrainer nevű, több-több kapcsolatot létrehozó kapcsolótábla. Lássuk a három tábla tartalmát összegyűrő lekérdezést:

```
--Kurzus szekció
SELECT
  1                as Tag,
  NULL             as Parent,
  Course.CourseID  as [Course!1!CourseID],
  Course.Title     as [Course!1!CourseTitle],
  NULL             as [Trainer!2!TrainerName]
FROM
  Course
WHERE
  Course.CourseID IN ('2073', '1913')
UNION ALL
--Trainers szekció
SELECT
  2,
```

```
  1,
  Course.CourseID,
  NULL,
  Trainer.LastName + ' ' + Trainer.FirstName
FROM
  Course
INNER JOIN
  CourseTrainer
ON
  Course.CourseID = CourseTrainer.CourseID
INNER JOIN
  Trainer
ON
  CourseTrainer.TrainerID = Trainer.TrainerID
WHERE
  Course.CourseID IN ('2073', '1913')
ORDER BY
  [Course!1!CourseID], [Trainer!2!TrainerName]
```

A kimenet:



Tag	Parent	Course!1!CourseID	Course!1!CourseTitle	Trainer!2!TrainerName
1	NULL	1913	Exchanging and Transforming Data Using XML and XSLT	NULL
2	1	1913	NULL	Soczó Zsolt
2	NULL	2073	Programming a Microsoft SQL Server 2000 Database	NULL
2	1	2073	NULL	Fóti Marcell
2	1	2073	NULL	Soczó Zsolt

A kétféle jellegű adathalmazt a UNION ALL operátorral kapcsoltuk össze. A Tag nevű oszlop tartalmazza azt, hogy az adott sor a hierarchia hányadik szintjén helyezkedik el. A két 1-es Tag-ű sor a hierarchia legfelső elemei, mindkettő egy tanfolyam címét és kódját írja le. Látható, hogy e sorok szülője NULL, ebből tudjuk, hogy ők a gyökérelemek. A 2-es Tag-el rendelkező sorok egy adott tanfolyam oktatóit reprezentálják. Ezek szülői az 1-es Tag-ű sorok, azaz a megfelelő tanfolyamok. Azokban a mezőkben, amelyek nem tartalmaznak értelmes információt *(például az oktatókat tartalmazó sorok tanfolyamcímet tartalmazó oszlopa)*, oda NULL-t generálunk, jelezve, hogy itt nincs értelmes információ. A lekérdezés végén látható ORDER BY azért fontos, hogy az adott tanfolyamhoz tartozó oktatókat mindig megtaláljuk közvetlenül a hozzátartozó tanfolyamsor után. Az oszlopoknak elég vad, de szisztematikus neveket adtam. A név első része a forrástábla neve, a felkiáltójel utáni második rész a hierarchia szintje, azaz a sorban található információhoz kapcsolódó Tag értéke, a harmadik pedig egy beszédes név az oszlop tartalmára utalva.

Ennek a kissé nehézkes, de következetes oszlopnévkódolásnak az az előnye, hogy a kapott eredményhalmazt egy megfelelően megírt értelmezőprogram felolvashatja, értelmezheti, és anélkül felépítheti az eredeti hierarchiát, hogy bármit is tudna a bejövő adatok szerkezetéről vagy értelméről. Azaz írhatunk egy rekurzív táblafelolvasó-fabejáró függvényt, amely elkezdi felolvasni a sorokat, és a Tag illetve Parent oszlopokban található értékek mentén navigálva kiolvassa a megfelelő oszlopokat, és azt megjeleníti, mondjuk egymásba ágyazott táblázatokban, vagy egy fára felfűzve.

Egy ilyen elven megírt program tetszőleges nagyságú és tartalmú, de ilyen struktúrában összeállított eredményhalmazt képes kicsomagolni. Ez a program már sokkal átláthatóbb lesz, mint a durrbele JOIN-nal generált tartalmat feldolgozó förmedvény, de még mindig problémás lesz a hierarchia különböző szintjein található információk formázott megjelenítése. Természetesen ekkor az eddig általános értelmezőprogramunkat el kell kezdeni specializálni, és bele kell drótozni, hogy például a 2-es szinten található sorokat *(ami az oktatókat tartalmazza)* nem egy táblázatban, hanem egy vesszővel elválasztott listában kellene megjeleníteni. A sorokat bejáró ADO kódok elkezdenek keveredni a <table>, <TR>, <TD>, <SPAN> és egyéb HTML formázó elemekkel, és újra kész a kódspagetti. Ismerős ez, ASP programozók? A kód fele Response.Write-tal van tele.

Pedig olyan szépen indult az egész! Olyan zseniális lett az a kis táblánk. Mit vethetünk még be, hogy az eredményhalmaz bejárását végző kódról elfeledkezhessünk, és csak az adatok formázására legyen gondunk? Nos, ha az eredményhalmaz valamilyen módon XML formátumú lenne, akkor XSLT segítségével könnyedén megformázhatnánk a forrásadatokat *(lásd XML cikksorozat)*. Hogyan tudunk ebből a szép kis táblából XML dokumentumot gyártani? Viszonylag egyszerűen. Felolvassunk az első sort. Látjuk, hogy a sor Parent mezője NULL, azaz ez egy gyökérelem lesz. Az elem neve legyen a forrástábla neve, amit az oszlop nevének első darabjából tudhatunk meg *(Course)*. Ennek megfelelően a generálandó XML tag valahogy így indul:

```
<Course
```

Ehhez az elemhez két értékes információ társul, a tanfolyam azonosítója és címe. Ez az info még mindig az első sorban található. Illesszük ezeket be a Course elembe, mint attribútumokat:

```
<Course   CourseID="1913"   CourseTitle="Exchanging
and Transforming Data Using XML and XSLT">
```

Mivel egyéb értékes adat már nincs ehhez a taghoz, lezárhatjuk, és továbbmehetünk a következő sorra.

A második sor Parent-je 1, ami nem más, mint az előbb megkezdett elem. Az ebben található információkat az előbb megnyitott Course elem gyermekelemeként vesszük fel, így reprezentálva a tábla által leírt hierarchiát. Az oktató nevét most is attribútumként tároljuk. A második sor feldolgozása után így néz ki a generált XML tartalom:

```
<Course   CourseID="1913"   CourseTitle="Exchanging
and Transforming Data Using XML and XSLT">
  <Trainer TrainerName="Soczó Zsolt"/>
```

A harmadik sor felolvasásával kiderül, hogy már nincs több oktató ehhez a tanfolyamhoz, mert újra egy 1-es Tag-ű sor jön, azaz egy újabb tanfolyam. Ekkor le kell zárnunk az első sornál megnyitott Course tagot, és elkezdeni egy következőt:

```
<Course   CourseID="1913"   CourseTitle="Exchanging
and Transforming Data Using XML and XSLT">
  <Trainer TrainerName="Soczó Zsolt"/>
</Course>
<Course
```

Innentől a Kedves Olvasó már könnyedén legenerálhatja papíron a maradék XML dokumentumtöredéket. Segítségképpen itt a puska:

```
<Course CourseID="1913" CourseTitle="Exchanging
and Transforming Data Using XML and XSLT">
  <Trainer TrainerName="Soczó Zsolt"/>
</Course>
<Course CourseID="2073" CourseTitle="Programming a
Microsoft SQL Server 2000 Database">
  <Trainer TrainerName="Fóti Marcell"/>
  <Trainer TrainerName="Soczó Zsolt"/>
</Course>
```

Ezt az itt végigjárt algoritmust bár elég munkás, de viszonylag egyszerű implementálni. De most tessék megkapaszkodni! Mi van, ha a korábbi lekérdezésünk mögé odabiggyesztünk egy FOR XML EXPLICIT záradékot?

```
...
ORDER BY
  [Course!1!CourseID], [Trainer!2!TrainerName]
FOR XML EXPLICIT
```

Nos, az egyoszlopos eredményhalmaz egy szem oszlopában egzaktul azt az XML dokumentumot kapjuk vissza, mint amit az előbb kézzel legeneráltunk. Viva SQL Server!

FOR XML EXPLICIT

Az SQL Server FOR XML EXPLICIT záradéka olyan szerkezetű táblát vár bemenetül, mint amit az előbb fabrikáltunk. Az SQL Server dokumentáció ezt univerzális táblának hívja, és pontosan úgy dolgozza fel a tartalmát, mint ahogyan az előbb mi is tettük. A példában a forrástáblák számunkra értékes információi attribútumként képeződtek le a cél XML adatfolyamba. Természetesen ez csak egyfajta XML reprezentáció, például a kurzusról szóló adatok lehetnének gyermekelemek is. Szerencsére ehhez is kapunk támogatást. Ha az oszlopnév mögé egy felkiáltójel után kíírjuk az element jelzőt, akkor az oszlop nem attribútumként, hanem gyerekelemként jelenik meg a kimenetben:

```
...
  Course.CourseID as [Course!1!CourseID],
  Course.Title    as
  [Course!1!CourseTitle!element],
...
```

Sokkal bonyolultabb struktúrák is létrehozhatók az XML EXPLICIT segítségével, ehhez további jelzőket kell megadnunk az oszlopok nevében. Részletek a Books Online-ban találhatók.

XML nézetek

A FOR XML EXPLICIT segítségével olyan XML reprezentációt hozhatunk létre, amilyet csak akarunk. Azonban 5-6 tábla felett egyre áttekinthetetlenebb lesz az univerzális táblát létrehozó lekérdezés. A komplexitást táblakimenetű függvények felhasználásával némileg csökkenthetjük, például a fenti lekérdezés hármas JOIN-ját berakhatnánk egy függvénybe.
A gyakorlatban azonban általában fordítva történik egy XML kimenetű lekérdezés tervezése. Kapunk egy XML sémát, ami leírja a generálandó XML dokumentum szerkezetét, és ehhez kell kitalálni az ennek megfelelő XML tartalmat generáló lekérdezést. Ekkor a séma alapján elő kell állítanunk az univerzális táblát, ami fárasztó, unalmas, és eléggé intuitív feladat. Mi volna, ha segítene valaki ebben? Például, ha a kapott XML sémába beírhatnánk a megfelelő SQL táblák neveit, mezőit, és ez alapján valami legenerálná nekünk az univerzális táblát.
Ki ez a láthatatlan segítő? ő az XSD annotált séma, vagy más néven XML nézet.
Az XSD az XML Shema Definition **[1]** rövidítése, amely 2001. március 31-e óta az aktuális XML sémaleíró ajánlás *(azaz még nem szabvány, de már nyugodtan építhetünk rá, legfrissebb változata a május 2-i)*. Segítségével le lehet írni tetszőleges, jól formázott XML dokumentum szerkezetét. Hasonló sémaleírás már régóta létezik a Microsoft házatáján, amelyet XDR-nek *(XML Data-Reduced)* hívnak. Amikor az SQL Server 2000 elkészült, akkor még nem volt XSD, csak XDR. Emiatt az XML nézeteinket is csak XDR annotált sémaként írhatjuk meg az SQL Server 2000 alatt. Aki éles környezetben tervezi XML nézetek használatát, az egyelőre maradjon is ennél. Azonban utólag letölthető **[2]** és feltelepíthető SQLXML2 kiegészítés az SQL Serverhez *(Beta 1 állapotú)*, amely már XSD annotált séma támogatást is tartalmaz.

Figyelem!
Bár a dokumentációban nem találtam rá utalást, de az XSD annotált sémák csak akkor működnek az SQL Serveren, ha a Microsoft XML Parser 4.0 *(Beta)* [3] is fel van telepítve a gépre.

Itt az ideje, hogy újra lássunk valami kézzelfoghatót. Írjunk egy XSD sémát, ami specifikálja tanfolyamok listáját tartalmazó XML dokumentumok szerkezetét **[4]**.

```
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="tanfolyamok"
  type="TanfolyamokTipus" />
 <xsd:complexType name="TanfolyamokTipus">
  <xsd:sequence>
  <xsd:element name="tanfolyam"
   type="TanfolyamTipus" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="TanfolyamTipus">
  <xsd:sequence>
   <xsd:element name="Cim" type="xsd:string"/>
   <xsd:element name="Oktato
    type="OktatoTipus"
    minOccurs="0"
    maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Kod" type="xsd:string"/>
 </xsd:complexType>
 <xsd:complexType name="OktatoTipus">
  <xsd:sequence>
   <xsd:element name="Nev" type="xsd:string"/>
   <xsd:element name="Email" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

Erre a sémára illeszkedik például az alábbi dokumentum **[4]**:

```
<?xml version="1.0" encoding="ISO8859-2"?>
<tanfolyamok>
<tanfolyam Kod="2063">
 <Cim>Introduction to ASP.NET</Cim>
 <Oktato>
  <Nev>Soczó Zsolt</Nev>
  <Email>zsolt.soczo@netacademia.net</Email>
 </Oktato>
</tanfolyam>
<tanfolyam Kod="2124">
 <Cim>Introduction to C# Programming</Cim>
 <Oktato>
  <Nev>Soczó Zsolt</Nev>
  <Email>zsolt.soczo@netacademia.net</Email>
 </Oktato>
<Oktato>
  <Nev>soci</Nev>
  <Email>soci@netacademia.net</Email>
 </Oktato>
</tanfolyam>
</tanfolyamok>
```

Ami ebből könnyen kihámozható: az összetett, több gyermekelemet vagy attribútumot tartalmazó dokumentumrészeket xsd:complexType elemekkel definiáljuk, amelyeken belül deklaráljuk a gyermekelemeit és attribútumait. Azaz kívülről befelé haladva, egyre inkább részletező módon írjuk le a dokumentum szerkezetét.
Hogyan házasítjuk össze ezt a sémát az SQL Server tábláival? Ehhez bele kell írogatnunk a séma egyes elemeibe az SQL Server táblákra és mezőkre utaló kifejezéseket *(innen az annotated schema elnevezés)*. A jelzéseinket egy külön névtérben hozzuk létre, így azok nem keverednek össze a séma eredeti elemeivel. A szükséges névtér az

```
xmlns:sql="urn:schemas-microsoft-com:mapping-schema"
```

amelyet – mint látható – általában az sql névtér rövidítéshez rendelünk. Ha véletlenül az eredeti sémában már használják az sql névtér alias-t, akkor semmi gond, egyszerűen más álnevet adunk a fenti urn-el azonosított névtérnek, és a későbbiekben mindenhol ezzel hivatkozunk rá az sql azonosító helyett.
A sémában szereplő elemek és az SQL Server táblák közötti kapcsolatot az sql:relation attribútum segítségével teremthetjük meg. Például a tanfolyamelemnek megfelelő SQL Server tábla a Course lesz. Ennek megfelelően a tanfolyamelemet így kell kiegészíteni:

```
<xsd:element name="tanfolyam"
 type="TanfolyamTipus" maxOccurs="unbounded"
 sql:relation="Course" />
```

Miután a tanfolyamelemnek megjelöltük az SQL Server tábla párját, a benne található elemeknek jelezni kellene, hogy melyik mezőkben találhatók meg a hozzájuk tartozó adat. Ebben az sql:field attribútum lesz a segítségünkre:

```
<xsd:element name="Cim" type="xsd:string"
 sql:field="Title" />
...
<xsd:attribute name="Kod" type="xsd:string"
 sql:field="CourseID" />
```

Az SQL táblát meghatározza a szülőelem *(tanfolyam)* sql:relation attribútuma, így azt a gyermekelemnél illetve az attribútumnál nem kötelező kiírni *(bár kiírhatjuk, ha úgy jobban átlátható a nézet)*.
Az Oktató elem már keményebb dió lesz. Ő a kapcsolótáblához, a CourseTrainer-hez tartozik, mert ahány bejegyzés van ebben a táblában *(ahányan oktatnak egy tanfolyamot)*, annyi Oktató elemet kell létrehozni, benne a trainer-ek adatait tartalmazó gyermekelemekkel. A probléma az, hogy honnan tudja a sémát feldolgozó processzor, az SQLXML2 ISAPI alkalmazás, hogy milyen kapcsolat van a Course és a CourseTrainer táblák között? Tőlünk, ha egy kicsit segítünk neki az alább látható módon:

```
<xsd:annotation>
 <xsd:appinfo>
  <sql:relationship name="CourseTrainers1"
   parent="Course"
   parent-key="CourseID"
   child="CourseTrainer"
   child-key="CourseID" />
 </xsd:appinfo>
</xsd:annotation>
```

Ezt a blokkot a séma elejére írjuk. Az XML kimenetet generáló alkalmazás *(XMLSQL2)* ebből tudja, hogy a két tábla a CourseID mezőkön keresztül kapcsolódik egymáshoz, így a szükséges kimenetet már elő tudja állítani egy egyszerű JOIN művelettel.
Az Oktató elemnél megadjuk a forrástáblát a már ismert sql:relation jelöléssel, és a kijelölt táblák közötti kapcsolatra pedig az sql:relationship attribútummal hivatkozunk, ahol az előbb létrehozott kapcsolat nevét adjuk meg:

```
<xsd:element name="Oktató
 type="OktatoTipus"
 minOccurs="0"
 maxOccurs="unbounded"
 sql:relation="CourseTrainer"
 sql:relationship="CourseTrainers1" />
```

Már csak egy lépés van hátra, amikor az oktatók neveit és e-mail címeit összekötjük a Trainer táblával. Ehhez egy újabb kapcsolatot kell kijelölnünk, amelyet a korábban látott módon az <xsd:appinfo> elembe veszünk fel:

```
<sql:relationship name="CourseTrainers2"
 parent="CourseTrainer"
 parent-key="TrainerID"
 child="Trainer"
 child-key="TrainerID" />
```

A kapcsolat a fantáziadús CourseTrainers2 nevet kapta. A név és e-mail mezőkben erre hivatkozunk:

```
    <xsd:element name="Email" type="xsd:string"
     sql:relation="Trainer"
     sql:relationship="CourseTrainers2"
     sql:field="Email" />
```

Az e-mail az Oktató elem gyermekeleme. A szülőelem más táblához *(a CourseTrainer-hez)* kapcsolódik, így minden gyerekelemnél, - az e-mail-nél is - meg kell adnunk a forrástábla, a kifejtett kapcsolat és a mező nevét is.
Minden elem megtalálta a párját az SQL Serveren, egyedül a tanfolyamok elem maradt árván. Ő egy állandó gyökérelem, amelynek az értéke nem függ az SQL Server adataitól. Az ilyen elemeket az sql:is-constant="1" kiegészítéssel különböztethetjük meg a többi, dinamikus tartalmú elemtől. Ennek hatására az elem változatlanul megjelenik a kimenetben. Nézzük meg az egész, készre annotált sémát is:

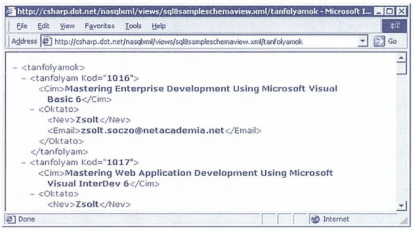
```
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:sql="urn:schemas-microsoft-com:mapping-s-
chema">
 <xsd:annotation>
  <xsd:appinfo>
   <sql:relationship name="CourseTrainers1"
    parent="Course"
    parent-key="CourseID"
    child="CourseTrainer"
    child-key="CourseID" />
   <sql:relationship name="CourseTrainers2"
    parent="CourseTrainer"
    parent-key="TrainerID"
    child="Trainer"
    child-key="TrainerID" />
  </xsd:appinfo>
 </xsd:annotation>
 <xsd:element name="tanfolyamok"
  type="TanfolyamokTipus" sql:is-constant="1"/>
 <xsd:complexType name="TanfolyamokTipus">
  <xsd:sequence>
  <xsd:element name="tanfolyam"
   type="TanfolyamTipus" maxOccurs="unbounded"
   sql:relation="Course" />
  </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="TanfolyamTipus">
  <xsd:sequence>
   <xsd:element name="Cim" type="xsd:string"
    sql:field="Title" />
   <xsd:element name="Oktato
    type="OktatoTipus"
    minOccurs="0"
    maxOccurs="unbounded"
    sql:relation="CourseTrainer"
    sql:relationship="CourseTrainers1" />
  </xsd:sequence>
  <xsd:attribute name="Kod" type="xsd:string"
   sql:field="CourseID" />
 </xsd:complexType>
 <xsd:complexType name="OktatoTipus" >
  <xsd:sequence>
   <xsd:element name="Nev" type="xsd:string"
    sql:relation="Trainer"
    sql:relationship="CourseTrainers2"
    sql:field="FirstName" />
   <xsd:element name="Email" type="xsd:string"
    sql:relation="Trainer"
    sql:relationship="CourseTrainers2"
    sql:field="Email" />
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Az XML nézet működés közben

Hogyan tudjuk kipróbálni az XML nézetünket?
Mindenekelőtt telepítsük fel az SQL2000 kiszolgálónkra az SQLXML 2 Beta 1-t **[2]**, és az MSXML Parser 4.0 Technology Preview-t **[3]**. Lehetőleg ne éles kiszolgáló legyen a célpont, mert abból bonyodalmak származhatnak.
A telepítés után megjelenik egy Microsoft SQL Server XML Tools mappa a Start menüben. Ezen belül a Configure IIS Support - Web Release 2 segítségével indul el az SQL Server XML támogatását konfiguráló konzol. Ebben a Default Web Site-ra állva jobb klatty, New Virtual Directory segítségével hozhatunk létre egy virtuális könyvárat a webkiszolgálón. Ezzel fogjuk elérni a SQL Servert HTTP protokollon keresztül.
A virtuális könyvtár legfontosabb jellemzői:

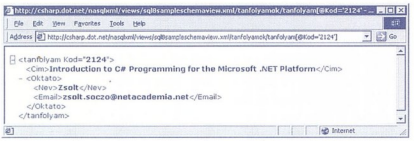
- Az ő neve, így fogunk rá hivatkozni az URL-ekben.
- Az SQL kiszolgálóhoz kapcsolódáshoz szükséges felhasználói azonosítás módja.
- Az adatforrás SQL Server neve és az elérni kívánt adatbázis.
- A HTTP-n keresztül engedélyezett műveletek. Ezek közül nekünk az XPath a legfontosabb, az kell az XML nézetünkhöz.
- A sémákat tartalmazó könyvtár elérési helye és virtuális neve. Létrehozunk egy schema típusú virtuális nevet, jelen esetben views néven és a Path mezőben található elérési útvonalat beállítjuk az XSD annotált sémáinkat tartalmazó könyvtárra.

Miután mindent sikeresen beállítottunk, és az annotált sémánk a fent beállított sémakönyvtárban van, már el is érhetjük a böngészőből:

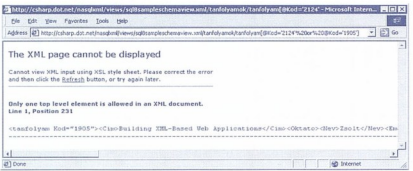


Az URL-ben az nasqlxml a virtuális könyvtár neve, a views a sémákhoz létrehozott virtuális név, az sql8sampleschemaview.xml pedig az előbb megírt annotált séma. De mit keres a /tanfolyamok az URL végén?
Az XML nézeteket szűrni lehet, a nézet neve után megadott XPath kifejezéssel *(lásd XMLGessünk sorozatunk)*. A /tanfolyamok XPath kifejezés azt jelenti, hogy kérem az összes

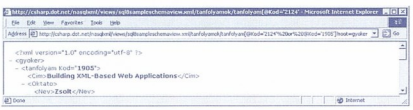
gyökérszinten található tanfolyamok elemet. Ilyen egy van, a gyökérelem, azaz megkapjuk a teljes dokumentumot, szűretlenül. És ha csak egy tanfolyam érdekel? Mi sem egyszerűbb:



Bonyolultabb kifejezéseket is megadhatunk, ám vigyázzunk, mert a kimenet egy XML dokumentum-töredék, ami nem biztos, hogy jól formázott lesz. Például lehet, hogy több mint egy gyökérelemet tartalmaz, amit nem tűr el az Internet Explorer sem:



Mivel gyakori, hogy csak töredékdokumentumot kapunk, jól jön az a lehetőség, hogy egy mesterséges gyökérelemet csempészhetünk be a legenerált XML dokumentum köré. Ennek érdekében a ?root=gyökérelemneve paramétert kell az URL végére írni:



Vajon hogyan generálja az XML nézetek adatait az SQL Server? Profiler-rel a színfalak mögé nézve gyorsan lebuktatható, hogy ő is bizony FOR XML EXPLICIT-et használ, aminek a forrásául szolgáló univerzális táblát *(lásd előző rész)* az SQLXML ISAPI alkalmazás állítja össze - helyettünk.
A választás rajtunk áll: vagy használjuk közvetlenül a FOR XML EXPLICIT-et, vagy annotált XSD *(esetleg XDR)* sémát írunk. Akinek ismeretlen a séma felépítése, annak bizonyára nagyon idegen és bonyolult az annotált séma készítése. Viszont aki ismeri, az többet nem fog bajlódni a FOR XML EXPLICIT univerzális táblájával.

XML sablonok *(XML Template)*

Akár FOR XML-lel, akár XML nézettel is generálunk XML adatokat, a könnyebb kezelhetőség illetve a belső részletek elfedése végett érdemes az adatforrásunkat XML sablon mögé rejteni.
Az XML sablonok felépítése nagyon egyszerű. Lássunk is mindjárt egy vázat:

```
<?xml version="1.0" encoding="ISO8859-2"?>
<courses
 xmlns:sql="urn:schemas-microsoft-com:xml-sql">
  <sql:header>
    <sql:param name="CourseIDs">1905</sql:param>
  </sql:header>
  <sql:query>
    exec webGetCourseXML @CourseIDs
  </sql:query>
</courses>
```

A sablon nem más, mint egy jól formázott XML dokumentum, amelyben az SQL Server specifikus elemeket az urn:schemas-microsoft-com:xml-sql névtérből származtatjuk. A <courses></courses> a sablon gyökérelemét képező xml elem, ez lesz a sablon által generált xml dokumentum gyökéreleme is.
A header részben adhatjuk meg a paraméterek listáját, amelyeket a sablonra hivatkozó URL-ekben adhatunk meg. Lehet alapértelmezett értéket is adni, így például a CourseIDs paraméter értéke 1905 lesz, ha a sablon hívásakor nem írjuk ki a paramétert.
A query elemek közötti részben kell megadni a sablon tartalmát adó xml adatforrást. Ez lehet egy XML kimenetű SQL lekérdezés, ahogyan az a példában is látszik. Közvetlen SQL parancsot is megadhatunk, de tárolt eljárásokat is használhatunk, mint a webGetCourseXML. Emellett XML sablont is megadhatunk adatforrásnak, és annak a kimenetét pedig a már látott módon XPath kifejezésekkel szűrhetjük meg. Például az előző fejezetben összerakott nézetünket csomagoljuk be egy sablonba:

```
<courses  xmlns:sql="urn:schemas-microsoft-com:xml-
sql">
  <sql:header>
    <sql:param name="CourseID">1905</sql:param>
  </sql:header>
<sql:xpath-query mapping-
schema="/nasqlxml/views/sql8sampleschemaview.xml">
  tanfolyamok/tanfolyam[@Kod = $CourseID]
  </sql:xpath-query>
</courses>
```

Ebben az esetben az adatforrást nem a query, hanem az xpath-query elemben adjuk meg, az XML nézetre a mapping-schema attribútumban hivatkozva. Az XPath szűrési feltételt az elem belsejébe kell írni.

Zárszó

Végignéztük a lefelé irányt, az adatok XML formátumban történő elérését. Egy későbbi alkalommal a visszafelé iránnyal is foglalkozunk, az updategram-ok kérdéskörével, amelyekkel közvetlenül XML adatokat juttathatunk el az SQL Serverre.
Aki szeretné alkalmazni az itt tanult tudást, az kérem lapozzon át az XMLGessünk cikkünkre, amely megmutatja a gyakorlati felhasználását az itt felépített XML adatforrásoknak.

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

A cikkben szereplő URL-ek:

[1]: XML Schema ajánlás http://www.w3.org/TR/xmlschema-0
[2]: SQLXML2 Beta 1 http://msdn.microsoft.com
/msdn-files/027/001/602/Search.asp
[3]: Microsoft XML Parser 4.0 Technology Preview
http://msdn.microsoft.com/xml/general/newinaprilre.asp
[4]: http://technet.netacademia.net/download/sql

XMLgessünk *(IV. rész)*

Bevezetés

Igazi csemegét van szerencsém bejelenteni. A kiszolgálóoldali HTML tartalomgenerálásnak nézünk a körmére, XSLT és adatbázis-alapokon. A cikk most párban jár az SQL sorozat cikkével, az egyik kiegészíti a másikat. Érdemes az SQL-lel kezdeni, és azután belevágni az XML finomságokba, itt és most.

A hagyományos ASP úton

Hogyan működik a tartalom generálása „hagyományos" ASP eszközökkel készített, adatbázisalapú webalkalmazásokban? ADO kapcsolatot nyitunk az adatbázis-kiszolgálóra, például az SQL Serverre. A kapcsolaton keresztül végrehajtunk egy SQL parancsot, amelyek eredményhalmazát ADO Recordset objektumként kapjuk meg. Ez azonban nem más, mint egy *(eléggé)* intelligens tömb, amelyből még elő kell állítanunk valamilyen HTML tartalmat. Ehhez végig kell lépkednünk egy ciklusban a Recordseten, és az aktuális adatbázissor mezőit sztringműveletekkel beleszerkesztjük a kimeneti adatfolyamba vagy sztringbe. Ez a gyalogosmódszer. Ezt csináljuk nap mint nap.

Amíg csak egy egyszerű lekérdezés kimenetét, például egy tábla tartalmát kell HTML-é alakítani, addig a „sztringhegesztős" megoldás elég átlátható tud maradni. Azonban a problémák akkor kezdődnek, amikor több táblából kell összeszedni a logikailag összetartozó információkat a megfelelő tartalom megjelenítéséhez. Ilyenkor teljesítmény szempontjából az lenne az előnyös, ha az összes megjelenítendő információt sikerülne egy eredményhalmazként visszakapnunk, egy hálózati körülfordulás alatt. Azonban ez általában nem szokott összejönni, vagy ha igen, akkor a generált eredményhalmaz olyan redundáns lesz, hogy a szükségesnél sokszor több információ fog utazni az adatbázis és a webalkalmazás között, mint amennyi kellene *(lásd párhuzamos SQL cikkünket)*. Ez akkor is nagy teljesítményveszteséget fog okozni, ha a webkiszolgáló és az adatbáziskiszolgáló egy és ugyanazon számítógépen vannak.

Ezért kerülőmegoldásokat kellett kidolgoznunk. Például igen gyakori feladat, amikor valamilyen objektumokat *(áruk, személyek, hírek, cikkek, ...)* ki kellett listázni, és minden tételhez szükség volt további részletező adatokra is - más táblákból. Ekkor használtuk azt az eljárást, hogy lekérdeztük az alapadatokat, a kapott Recordsetből elkezdtük felépíteni a listát, és amikor az adott tétel részletes adataira volt szükség, akkor egy, a tétel azonosítójával paraméterezett lekérdezéssel lehívtuk az adott elem további tulajdonságait. Ez azt jelentette, hogy ahány tétel volt a listában annyi plusz lekérdezést kellett végrehajtani minden egyes weboldal végrehajtásakor. Ennél jobban nehéz pazarolni az erőforrásokat *(na jó, a memory leak azért durvább :)*.

Milyen „szakos" megoldást kaptunk az ASP alapú rendszerekben erre rendkívül gyakori és triviális problémakörre? Semmilyet. Trükkös táblákat építhetünk JOIN és UNION operátorok segítségével, ám ezeknek nehéz kiolvasni, illetve megjeleníteni a tartalmukat. Jött a szokásos kód – HTML elemek – kód – HTML elemek spagetti. Karbantarthatatlan, lassú, ronda.

Amíg nincs ASP.NET-ünk, ami adatkötéssel bíró kiszolgálóoldali vezérlőkkel *(részben)* megoldaná a problémáinkat, addig is érdemes fontolóra venni az XML-XSL-ben rejlő lehetőségeket. Ez nem azt jelenti, hogy a .NET után nem lesz létjogosultsága az XSL alapú megoldásoknak, csak azt, hogy abban már kapunk nem XML alapon is olyan eszközöket, amelyek segítségével viszonylag könnyen tudunk adatbázisalapú adatokat megjeleníteni. Legalább lesz alternatívánk. A kétféle megközelítés összehasonlításáról az **[1]** címen lehet bővebben olvasni.

Maradjunk a végleges eszközöknél, és nézzük meg, hogy XSLT segítségével hogyan lehet az előbb vázolt listázási problémát megoldani. Az alapfelállás: van egy XML adatforrásunk, annak kimenetét szeretnénk XSLT-vel HTML-lé konvertálni. Adatforrásként ideális választás az SQL Server, erről bővebben ugyanezen szám SQL cikkében olvashat a Kedves Olvasó.

A konverzió történhet ügyféloldalon *(erről már részben értekeztem a cikksorozat előző részében)*, illetve az XML-t közvetlenül nem kedvelő böngészők esetén kiszolgálóoldalon. Most az utóbbival ismerkedünk meg.

Belép az XML

Nézzük először az adatforrás kérdését. Ami általában szóba jöhet:

1. Közönséges XML állomány – ritkán van haszna, de például konstans táblázatok, legördülő listák forrásaként jó lehet
2. Közvetlen SQL Server kapcsolat SELECT ... FOR XML... paranccsal, ADO eléréssel
3. SQL Server XML sablon közvetlen XML DOM eléréssel
4. SQL Server XML nézet XML DOM feldolgozással
5. ..., például Exchange 2000, SharePoint Portal Server

Jelen cikkünkben a második és a harmadik megoldást nézzük meg részletesen, de minimális eltéréssel a negyedikre is érvényes lesz az ott leírt elmélet.

Közvetlen adatbázislekérdezés

A megoldás gondolatmenetét és a hozzá kapcsolódó JScript kódot láthatjuk a következő pontokban.

1. ADO kapcsolatot nyitunk az SQL Serverre. A hibákat try-catch blokkban kezeljük le.

```
try
{
    var conNetAcademia =
        Server.CreateObject("ADODB.Connection");
    conNetAcademia.CursorLocation = adUseClient;
    conNetAcademia.Open("Provider=SQLOLEDB;"
    + "Initial Catalog=Netacademia;"
    + "Data Source=(local);"
    + "Integrated Security=SSPI;");
```

2. Létrehozunk egy ADO parancsobjektumot az SQL lekérdezés végrehajtásához.

```
var cmd =
Server.CreateObject("ADODB.Command");
cmd.ActiveConnection = conNetAcademia;
cmd.CommandText = strSQL;
```

3. Létrehozunk és megnyitunk egy ADODB.Stream objektumot.

```
var XMLStream =
Server.CreateObject("ADODB.Stream");
XMLStream.Open();
```

4. Végrehajtjuk az XML tartalmat generáló parancsunkat, az eredményt a Stream-be irányítjuk.

```
cmd.Properties("Output Stream") = XMLStream;
cmd.Execute(null, null, adExecuteStream);
```

5. Lezárjuk az adatbáziskapcsolatot, a Stream tartalma attól még megmarad.

```
conNetAcademia.Close();
```

6. Létrehozunk egy DOMDocument objektumot. Ebbe kerül bele a lekérdezés által generált XML tartalom.

```
var DomCourse =
Server.CreateObject("MSXML2.DomDocument");
DomCourse.async = false;
```

7. A példában a lekérdezés eredményeként olyan XML dokumentumtöredéket kapunk vissza, amelynek több mint egy gyökér eleme van. Ezért be kell illesszünk egy „mesterséges" gyökeret, hogy jól formázott XML dokumentumot kapjunk.

```
var strXML =
'<?xml version="1.0" encoding="ISO8859-2"?>'
+ '<courses>'
+ XMLStream.ReadText(adReadAll)
+ "</courses>";
```

8. Mehet a DOMDocument-be a Stream tartalma. Miután a loadXML metódus lefutott, a Stream tartalmára már nincs szükség, hisz a DOM-nak már van másolata az XML dokumentumból. Azaz lezárhatjuk a Stream-et.

```
DomCourse.loadXML(strXML);
XMLStream.Close();
```

9. Az első kódblokkunknak vége, a hibakezelő blokkunk is véget ér. Hiba esetén a ReportParseError és a GeneralErrorFormatter függvényekben reagáljuk le a hibaeseményeket. Ezek forrása a **[2]** címen található.

```
GeneralErrorFormatter(exception);
if (DomCourse != null)
{
    ReportParseError(DomCourse.parseError);
}
Response.End();
```

10. Létrehozunk egy másik DOMDocument-et a stíluslap részére, és betöltjük a stíluslapot. Mivel ő is egy szabályos XML dokumentum, a DOMDocument szívesen magába fogadja.

```
try
{
    var DomXSL =
    Server.CreateObject("MSXML2.DomDocument");
    DomXSL.async = false;
    DomXSL.load(strXSLPath);
}
catch(exception)
{
    if (DomXSL != null)
    {
        ReportParseError(DomCourse.parseError);
        Response.End();
    }
    GeneralErrorFormatter(exception);
    Response.End();
}
```

11. A transzformáció előtt be kell állítani a kimenet karakterkódolását *(Central European)*, különben a böngésző nem fogja tudni milyen nyelvű a generált HTML dokumentum.

```
Response.Charset = "ISO8859-2";
```

12. Végrehajtjuk a transzformációt, a kimenetet közvetlenül a Response objektumba irányítva *(hatékonyság!)*. A transzformációt az XML forrást tartalmazó dokumentumobjektumon hajtjuk vége, paraméterként a stíluslap dokumentumot és a kimeneti Stream-et, a Response-ot adjuk meg. Kihasználjuk, hogy az ASP Response objektum implementálja az IStream COM interfészt, így a DOM tudja, hogyan kell vele kommunikálni. A Response-ba nyugodtan írhatunk egyéb tartalmat a transzformáció előtt és után is. Azok összefűződnek az itt beírt tartalommal.

```
DomCourse.transformNodeToObject(DomXSL,
                                Response)
```

13. Lezárjuk a hibakezelő kódot.

```
}
catch(exception)
{
    GeneralErrorFormatter(exception);
}
```

A teljes kód gazdagon kommentezve a **[2]** címen érhető el, ahol ki is lehet próbálni élő adatokkal.

Adatelérés XML sablonon keresztül

Az előbbi példa talán legbonyolultabb része az adatbázist lekérdező kód volt. Mivel az SQL Server XML formátumú kimenetét minden további nélkül elérhetjük közvetlenül HTTP protokollon keresztül is, valamint kihasználva, hogy a DOMDocument load metódusa URL-t is elfogad forrásként, jelentősen leegyszerűsíthető a forrásadatok megszerzésének folyamata.

Az SQL Server XML sablonok illetve XML nézetek segítségével képes XML tartalmat HTTP-n keresztül szolgáltatni. Az XML sablon mögött vagy egy FOR XML... végű SELECT SQL lekérdezés áll, vagy egy XML nézet. Az XML nézet közvetlenül is megcímezhető XPath kifejezés segítségével. Részletek az SQL cikkünkben.

Az adatbázisháttér részletei jelen pillanatban nem fontosak, a lényeg, hogy van egy olyan URL-ünk, amiről XML formátumú adatok érkeznek. Erről az URL-ről az XML DOM közvetlenül le tudja tölteni az XML forrásdokumentumot, így elfelejthetjük az ADO objektumokat. Ennek megfelelően nézzük meg, hogyan változik a programunk első része.

1. Definiáljuk a forrásainkat. Az első változóba töltött URL egy *(az online verzióban paraméterezhető)* XML sablonra mutat.

```
var strXMLPath = "http://"
+ Request.ServerVariables("SERVER_NAME")
+ "/nasqlxml/template/course.xml";
var strXSLPath =
    Server.MapPath("course.xsl");
```

2. Létrehozunk egy XML DOMDocument objektumot. Nagyon fontos látnunk, hogy most speciális helyzetben fogjuk használni a DOM-ot: webalkalmazásban futtatva egy webkiszolgálón *(akár önmagán)* található HTTP erőforrást fogunk elérni. Erre fel kell készíteni a dokumentumobjektumunkat, mivel ez speciális igénybevételt jelent neki. Ehhez a ServerHTTPRequest tulajdonságát kell beállítani true-ra, így a háttérben nem az XMLHTTP, hanem a ServerXMLHTTP objektum tölti le az URL-ről az XML adatokat. Csak ez működik kiszolgálóoldalon!

```
var DomCourse =
Server.CreateObject("MSXML2.DomDocument");
DomCourse.async = false;
//!!!
DomCourse.setProperty("ServerHTTPRequest",
                      true);
DomCourse.load(strXMLPath);
```

Ettől a ponttól kezdve minden ugyanaz, mint az előző példában, azaz XSL betöltés majd transzformáció következik. 4 sorral megúsztuk az XML adatok betöltését!

Gondolhatnánk, hogy a járulékos réteg, a webkiszolgáló megjelenése az adatfolyamban jelentősen lassítja a lap generálását. A méréseim alapján azonban gyakorlatilag nem volt különbség a két módszer között. Ez különösen akkor igaz, ha az SQL Server plusz a HTTP eléréshez szükséges IIS és a valódi webkiszolgáló külön gépen vannak.

XSLT Cache alkalmazása

Az előbbi két példában volt egy közös pont, ami jelentős teljesítményveszteséget okoz egy nagyterheltségű webalkalmazásban. Ez pedig az XSL dokumentum betöltése és lefordítása. A legtöbb esetben a stíluslapok eléggé állandóak, ritkán módosítják őket. Ennek ellenére mi minden egyes lap legenerálásakor betöltjük és lefordíttatjuk az XSL dokumentumot. Nem lehetne valahogyan globálisan, minden lap számára látható módon előkészíteni egy XSL-t tartalmazó objektumot, és azt minden lapban felhasználni? A Microsoft XML DOM implementációja ehhez hathatós segítséget nyújt. Milyen központi helyünk van a közös objektumok tárolására? Természetesen a global.asa. Ott az Application_OnStart eseményben létrehozhatunk olyan alkalmazásszintű objektumokat, amelyek minden lap számára láthatóak lesznek.

Az alkalmazásszinten létrehozott objektumokkal szemben igen erős követelmények vannak, mert egyszerre sok lap, sok thread használhatja őket. Az eddig használt DOMDocument objektumunk nem állná ezt a rohamot *(mint ahogy az összes VB6-ban írt objektumunk sem!)*, helyette a FreeThreadedDOMDocument komponensből kell egy példányt létrehoznunk. Ő free-threaded *(Multi Threaded Apartment)* modellű objektum, amely fel van készítve a párhuzamos felhasználásra.

Mivel a legtöbb időt nem is az XSL fizikai betöltése, hanem a lefordítása jelenti, ezért a Microsoft XML DOM-ban létrehoztak egy XSLTemplate nevű egyedet, aki lefordított XSL stíluslapot képes tárolni. A többszálú felhasználás miatt nem is tárol szálhoz kapcsolódó adatokat. Azonban a konkrét transzformációk végrehajtása során kellene állapotinformációkat is tárolni, például a későbbiekben részletezett paramétereket. Mivel párhuzamosan több transzformáció is működhet a webkiszolgálón *(ne felejtsük, az IIS többszálú)*, így vagy az XSLTemplate-nek kellene megjegyezni, hogy melyik hívó szál transzformációja hányadán áll, vagy létrehozunk egy másik objektumot immár nem alkalmazás, hanem lapszinten, így abban már tárolhatunk lap *(szál)* függő információkat. Az első megoldás kevesebb memóriafelhasználással jár, de sok szál esetén lassúvá válna a szálak közötti szinkronizáció miatt. A második megoldás több memóriát igényel, hisz annyi másolatobjektumot kell létrehozni, ahány lap fut egyszerre. A Microsoft – mint a legtöbb esetben – nagyon helyesen a sebesség mellett döntött.

Az XSLTemplate-nek van egy testvére, az XSLProcessor. Amikor konkrétan transzformálni szeretnénk egy weblapban, akkor a globálisan eltárolt XSLTemplate createProcessor metódusának meghívásával kapunk egy XSLProcessor objektumot. Ez már könnyedén áttranszformálja a korábban látott módszerek valamelyikével felépített forrás DOMDocument objektumunkat a transform metódus hívásával.

Ültessük át a gyakorlatba az eddigi monoton elméletet! Nézzük át lépésenként, mi a teendőnk a global.asa-ban.

1. A webalkalmazás elindulásakor létrehozunk egy XSLTemplate objektumpéldányt.

```
<SCRIPT LANGUAGE=JScript RUNAT=Server>
function Application_OnStart()
{
    var XSLTemplate =
    new ActiveXObject("MSXML2.XSLTemplate");
```

2. Létrehozunk egy FreeThreadedDOMDocument objektumot az XSL stíluslap betöltéséhez, és betöltjük az XSL-t tartalmazó állományt.

```
var xslDoc =
new ActiveXObject(
    "MSXML2.FreeThreadedDOMDocument");
xslDoc.async = false;
xslDoc.load(Server.MapPath("course.xsl"));
```

3. Elkészítjük az XSLTemplate-et. A forrása természetesen a már betöltött stíluslapunk lesz.

```
XSLTemplate.stylesheet = xslDoc;
```

4. Az DOMDocument elvégezte a dolgát, mehet. A sablonnak már megvan a lefordított XSL, nincs szükség többé a forrásra.

```
xslDoc = null;
```

5. Tároljuk el a sablont egy alkalmazásszintű változóba, hogy a lapok felhasználhassák. Ezzel vége is az eseménykezelőnek, zárjuk le a függvényt és a scriptblokkot is.

```
    Application("XSLTemplate") = XSLTemplate;
}
</SCRIPT>
```

Ennyi a global.asa története. Most lássuk, mit kell tennünk a weboldalakban a sablon használatához.
Az első példánk 11. lépéséig, azaz a karakterkészlet beállításáig minden ugyanaz, azaz megszerezzük a forrás XML adatokat egy DOMDocument objektumba. Ezután fog különbözni a feldolgozás.

1. Létrehozunk saját XSLProcessor példányt a globálisan előállított sablon createProcessor metódusával.

```
xslProc =
Application("XSLTemplate").createProcessor();
```

2. A processzor input paraméterében átadjuk a transzformálandó forrásobjektumunkat.

```
xslProc.input = DomCourse;
```

3. Végrehatjuk a transzformációt. A végeredményt nem kapjuk meg visszatérési értékként, hanem az output tulajdonságon keresztül férhetünk hozzá.

```
xslProc.transform();
Response.Write(xslProc.output);
```

A szenzációs az ebben a sablonobjektumot alkalmazó módszerben, hogy ez a legegyszerűbb és a leggyorsabb az eddigi példáim közül. Egy felhasználó esetén persze nem sokat nyerünk, de egy normális webalkalmazást általában nem egy ember használ *(ha igen, akkor úgyis bezár :))*

Paraméterek átadása az XSLT-nek

Sokszor feladat az *(lásd a cikksorozatunk előző része)*, hogy egy adott XSLT transzformációba egy-két változót kellene becsempészni, amit a konkrét meghíváskor állítunk be.
Semmi probléma. Nem véletlenül fejtegettem a threading modelleket az XSLTemplate és a XSLProcessornál. Az XSLProcessor rental-threaded, így neki lehetnek szálhoz kötött adatait, nem véletlen, hogy minden lap külön példányt hoz létre belőle. S ha már laponként egyedi a processzor, miért ne lehetne neki paramétereket átadni, amelyek megjelennének a transzformációnkban?
Az XSLT nyelvtana megengedi paraméterek definiálását, amelyeket az XSLT processzornak futásidőben adunk át. A paramétereket a XSLT-ben az <xsl:param> elemmel lehet létrehozni:

```
<xsl:param name="sorrend" />
```

Értéket az XSLT-ben nem adunk neki, majd az XSLT processzoron keresztül tesszük ezt. Hogyan hivatkozhatunk az XSLT-ben a paraméterre? A nevével, egy $ jelet írva eléje. Például, ha egy XSLT-vel történő sorbarendezésnél a sorbarendezés irányát szeretnénk paraméterrel megadni, akkor azt a következőképpen fejezzük ki:

```
<xsl:sort select="" order={$sorrend} />
```

A paramétert ebben az esetben <xsl:stylesheet ...>-en belül hozzuk létre globálisan.
Jöhet az XSLProcessor. Van neki egy addParameter metódusa, amelyben meg kell adni a behelyettesítendő paraméter nevét és értékét. Ezt behelyettesíti az XSLT-be, ami után végrehajthatjuk a transzformációt.

```
xslProc. addParameter("sorrend", "descending");
```

Nyilván amennyi paraméterünk van, annyiszor kell alkalmaznunk a metódust.

Zárszó

Az XML technológiák ismertetése nagyon papírigényes, ezért a működő példáknak csak a tizede látható az újságban, papíralapon. A virtuális esőerdőt azonban nem kíméltem, így aki közelebbről is szeretné megnézni működés közben, teljes szépségében a példákat, kérem látogasson el a **[2]**-es címre.
Nyáron az XMLGessünk is elmegy hűvösebbre, de ez nem jelenti azt, hogy hűvösre tettük a témát, csak azt, hogy pihen egy kicsit. De ősszel újult erővel, akkor már *(mi más, ha nem)* .NET alapokon fogunk XMLGetni.

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

A cikkben szereplő URL-ek:

[1]: A Practical Comparison of XSLT and ASP.NET
http://msdn.microsoft.com/library/Welcome/dsmsdn/xml02192001.htm
[2]: http://technet.netacademia.net/feladatok/xml

Aktív és passzív FTP

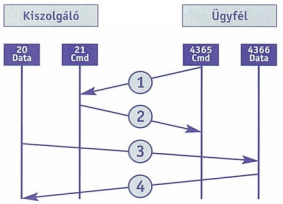
Gyakran *(főleg tűzfalak konfigurálása során)* felmerül az a kérdés, hogy mi a különbség az aktív és a passzív FTP kommunikáció között?

Párosan szép az élet...

Az FTP érdekes állat: amellett, hogy tisztán TCP alapon működik, azok közé a protokollok közé tartozik, amelyek egy-egy kommunikáció során nem egy, hanem két csatornát tartanak nyitva. Ez a két csatorna a parancs *(command vagy control)* és az adat *(data)* csatorna. A paranccsatorna a kliens dinamikus portjáról a kiszolgáló 21-es *(FTP)* portjára nyílik, míg az adatcsatorna a ügyfélgép egy másik *(de továbbra is az ftp kliensprogram által kézbentartott)* portja és a kiszolgáló 20-as *(FTP-DATA)* portja között épül fel – legalábbis az esetek egyik részében. Az aktív és passzív FTP közötti különbség pontosan az adatcsatorna hollétében van.

Aktív FTP

Aktív FTP esetén az ügyfélalkalmazás egy dinamikus portról *(pl. 4365)* a kiszolgáló 21-es portjára csatlakozva felépíti a parancscsatornát. Az adatcsatorna felépítését a kiszolgáló kezdeményezi, mégpedig a *(kiszolgáló)* 20-as portjáról a kliensalkalmazás által kézben tartott következő szabad dinamikus portra *(pl. a 4366-ra)*. Ez utóbbi portcím a további kommunikáció során mindig változik, azaz az egyes adatcsatornák mindig más dinamikus portra érkeznek *(de mindig a kiszolgáló 20-as portjáról indulnak)*, valahogy így:



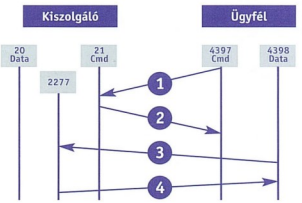
Aktív FTP

A bejelentkezéskor csupán a 21-es portra csatlakozunk, míg a dir parancs kiadásakor új csatorna jön létre, a kiszolgáló 20-as és a kliens 4366-os portja között.
Az aktív FTP-ben az adatcsatornát nem a kliens építi fel: ő csak „kinyitja" a következő portot, annak címét elküldi a kiszolgálónak *(PORT parancs a paranccsatornán)*, és várja a csatlakozást. Pontosan ez a probléma: az adatcsatorna felépítését a kiszolgáló kezdeményezi, ezt pedig a tűzfalak többsége érthető biztonsági okokból nem engedi.

Egy megoldás van azért: ha a tűzfal „okos", és figyeli az FTP kommunikáció tartalmát, kiolvashatja a kiszolgáló részére elküldött portcímet, és ebben az egyetlen esetben engedélyezheti a kapcsolatfelvételt.

Passzív FTP

A kiszolgálóoldali kapcsolatfelvétel elkerülése érdekében találták ki a passzív FTP-t. Ebben az esetben, miután a kliens felépítette a paranccsatornát *(a 21-es portra)*, PORT parancs helyett PASV parancsot küld át rajta, a kiszolgálót ezzel átkapcsolva passzív üzemmódba. A kiszolgáló erre megnyit egy portot *(példánkban a 2277-est)* a leendő adatcsatorna részére, és most ő küldi vissza az újonnan nyitott port címét *(a PORT paranccsal)* az ügyfélnek, aki ezután felépíti a második kapcsolatot, immár a két dinamikus port között. A dinamikus portcímek persze változhatnak, és az egyes adatcsatornák most is mindig új portcímre épülnek ki.



Passzív FTP

Az ügyfélprogramok többsége csak a passzív FTP-t támogatja, bár az Internet Explorer-nél külön be kell kapcsolni, ha használni szeretnénk. A Tools / Internet Options párbeszédablak Advanced oldalán jelöljük be az „Use Passive FTP for compatibility with some firewalls and DSL modems" sort.
Míg a passzív FTP a kliensoldali problémákat megoldja, a kiszolgálóoldalon újabbakat okozhat: hiszen ilyenkor a kiszolgálónak kell minden egyes látogató részére adatcsatornaportokat nyitogatnia, és a portok bizony egy idő után elfogyhatnak. A megoldás az intelligens tűzfalak és az aktív FTP használata lehet.

Fülöp Miklós
mick@netacademia.net

A mikroprocesszorok hatása

Középiskolás voltam, amikor először olvastam egy cikkben - amit barátom, Paul Allen talált az Electronics magazinban - a mikroprocesszorról. Paul és én ekkor már írtunk szoftvereket, de ez a cikk ráébresztett arra, hogy pénzt is kereshetnénk ezzel. Akkor még nem voltak olyan cégek, amelyek kizárólag szoftverfejlesztéssel foglalkoztak, a számítógépgyártók *(például IBM és Digital)* saját egyedi szoftvereket készítettek gépeikhez. Természetesen nem akarták, hogy Paul vagy én készítsük el ezeket.

Az Intel mikroprocesszorai ezt a helyzetet egy csapásra megváltoztatták. Először is, az Intel nem gyártott számítógépet vagy szoftvert, bár ahhoz, hogy bármit kezdeni lehessen vele, új chip-jükhöz szoftverre volt szükség, így Paul és én lehetségesnek láttuk egy szoftvercég beindítását. Másodszor, világosan látszott, hogy a mikroprocesszor a gyártási költségek csökkentésével, és az új képességek megvalósításával teljesen átformálja a számítógépipart.

Az Intel új chip-jének hirdetése, az „Integrált elektronika új korszakának bemutatása" igen előremutatónak bizonyult.

Amikor Ted Hoff *(Federico Faggin-nal és Stan Mazor-ral közösen)* az Intel-nél kifejlesztette a mikroprocesszort, nem gondolta, hogy megváltoztatja a világot. Amikor az Intel egyik vásárlója azt kérte, hogy számológépekhez készítsenek vagy egy tucat egyedi chip-et, Hoff úgy gondolta, hogy egyszerűbb egyetlen általános célú chip-et gyártani, amely majd a szoftver segítségével képes lesz ellátni a különböző feladatokat. Ennek eredménye lett az Intel 4004 mikroprocesszor, ami szilíciumdarabkára épített 2300 apró tranzisztorból állt, és tulajdonképpen egy számítógép volt, egyetlen chip-en. 1971-ben mutatták be, és egy egész iparágat alapoztak meg vele.

Nem egészen PC

A mikroprocesszor megváltoztatta az én életemet is, annak segítségével, ami a „Világ első miniszámítógép készlete" – a MITS Altair 8800 - lett. A Tiltott bolygó *(Forbidden Planet)* című film egyik világa alapján elnevezett, 1975-ben kiadott Altair egy Intel 8080-as mikroprocesszorra épült, és kevesebb mint 400 dollárba került, de nem lehetett semmi hasznosat tenni vele. Nem volt billentyűzete, képernyője és nem volt szoftvere sem, ezért Paul és én létrehoztunk egy társulást, amit Micro-Soft-nak hívtunk, és írtunk néhány szoftvert, ami képessé tette az Altair-t néhány egyszerű feladat elvégzésére. Hat évvel később, amikor az IBM kiadta az első modern PC-t, mi készítettük hozzá az operációs rendszert.

A mikroprocesszorra épülő PC-k megváltoztatták a világot. Forradalmasították információink összegyűjtését, tárolását és felhasználását, a kommunikációt, a munkát, a tanulást és a játékot. Húsz éve még senki nem használt számítógépet, csak ha ez volt a hobbija, vagy a munkája. A mai PC-k fejlett szoftverei és több mint 9 millió tranzisztort tartalmazó mikroprocesszorai segítségével még egy gyereknek is rendelkezésére állhat egy régi mainframe gépénél nagyobb számítási teljesítmény.

Egy olcsó otthoni PC-vel vezethetjük a házi költségvetésünket, elkészíthetjük adóbevallásunkat, levelezhetünk a barátainkkal, CD-t és rádiót hallgathatunk, híreket olvashatunk, tanácsot kérhetünk orvosunktól, játszhatunk, elintézhetjük nyaralásunkat, megnézhetünk egy házat, vagy akár autót is vehetünk... a lista végtelen.

A mindenütt megjelenő chip

Manapság a mikroprocesszorok mindenütt jelen vannak: mobiltelefonokban, CD lejátszókban, autókban, de legnagyobb hatása a PC-kben való alkalmazásuknak van.

A PC és az Internet együttesének segítségével az információk és hírek gyorsabban és szabadabban jutnak el rendeltetési helyükre, segítve ezzel a zárt gazdaságok megnyitását, és az elnyomott nemzetek demokratizálódását, mert a kibertérben nem lehet határokat építeni *(Dehogynem! :) - a ford.)*. A mikroprocesszorra épülő PC-k több szabadságot biztosítanak az embereknek, és lehetővé teszik, hogy kezdjenek valamit a szabadságukkal. Nem meglepő, hogy az idén 30 milliárd mikroprocesszort adnak el, és jövőre már lehet, hogy több PC-t vesznek, mint TV-t.

Tessék kapaszkodni!

Még csak a digitális kor küszöbén járunk. Az elmúlt két évtizedben a mikroprocesszorok teljesítménye több mint egymilliószorosára nőtt, de lehet, hogy ez a következő húsz évben megismétlődik, és a processzorokban talán majd egymilliárd tranzisztor lesz.

A következő években még biztos a PC lesz a fő számítástechnikai eszköz, de hozzá lesznek kapcsolva olyan mikroprocesszorra épülő eszközök, melyek még egyszerűbbé teszik életünket, és tudni fogják, mikor milyen információra van szükségünk.

Egy valóban intelligens ház

Otthon majd a hangunkkal fogjuk irányítani a PC-nket, ami automatikusan mentést készít az információinkról, frissíti a szoftverét, és szinkronizálja magát a TV-vel, a mobiltelefonnal, a kézi PC-vel és az otthoni hálózatunk összes eszközével. A hűtőgépünk tudni fogja, hogy mennyi étel van benne, és recepteket fog ajánlani. A TV-nk segítségével megrendelhetjük majd a reklámban látott terméket, ha pedig éppen nem lesz kedvünk TV-t nézni, olvashatjuk elektronikus könyvünket, ami tudni fogja, hogy kik a kedvenc szerzőink, és automatikusan letölti legújabb műveiket. Ha úgy döntünk, hogy ezek egyikét elolvassuk, az árával megterhelődik a bankszámlánk.

Úgy hangzik, mint egy fantasztikus regény? Ez néhány éve még tényleg így lett volna, de a mikroprocesszornak, valamint a szoftverek, a hardverek, az Internet és a telekommunikáció hihetetlen fejlődésének köszönhetően minden amit leírtam, megvalósítható.

Bill Gates

A „**tech.net magazin Brainstorm**" a Dupla KV rovathoz hasonló, ám a személyes kérdésfelvetést és vitát is lehetővé tevő rendezvény, melynek célja:

- **az elsőre talán ismeretlen technológiák élő bemutatása**
- **a cikkekhez kapcsolódó kódok megírása/kipróbálása**
- **a terjedelmi okokból kimaradt információk átadása**

E magazinnal együtt a rendezvényre érvényes belépőjegyet minden előfizetőnkhöz eljuttattuk.
További információk a belépőjegyen olvashatók.

Várjuk Önöket a NetAcademia Mesterkurzusokon

NETACADEMIA

A legjobbakat tanítjuk.

(Bár belépőjegyet adtunk, ez a tény önmagában nem biztosítja helyét a rendezvényen. A jegy célja előfizetőink elsődlegességének biztosítása, de a korlátozott résztvevői létszám (100 fő) miatt a regisztráció kötelező. Jelentkezzen, amíg nem késő!)

http://technet.netacademia.net/brainstorm

HunNet RL512W

512 kbit/sec-os szimmetrikus forgalomfüggetlen csatlakozás az internetre havi 35.000 Ft-ért

.hu Hungarian Networks

Tel: 06-40-hunnet (486-638)


(no limits)
SQL
tanfolyamok
Tanfolyamkód: 2072
Tanfolyamkód: 2073
Tanfolyamkód: 1609
Security
Tanfolyamkód: 2159
Tanfolyamkód: 1561
Tanfolyamkód: 2150
Windows 2000
tanfolyamok
Tanfolyamkód: 2087
Tanfolyamkód: 1560
Tanfolyamkód: 2203
A tanfolyamkódok megfejtéséért és további információért látogasson el honlapunkra:
http://www.netacademia.net
Egyedi tanfolyamok
Tanfolyamkódok: WN-1, WN-2, WN-3, NJB-4, NJB-6
.net
tanfolyamok
Tanfolyamkódok: 2063, 2124 2415, 2349
NETACADEMIA
A legjobbakat tanítjuk.

(folyt. köv.)