tech.net

WORKING WITH WINDOWS

III. évfolyam
8-9. szám

Ára: 2688 Ft

A TARTALOMBÓL

ISSN 15865185
9 771586 518005 08

A LEGENDÁS ÖT KILENCES. KÖZELEBB VAN MINT GONDOLNÁ!

A szerver operációs rendszereknél, az a bizonyos öt kilences a megbízhatóság legfelsőbb fokát jelzi. Gyakorlati nyelvre lefordítva, ez a teljes rendelkezésre állás mellett, mindössze 5 perc kiesést jelent évente!* A 99,999%-os megbízhatósági mutató már nem csak laboratóriumi körülmények között érhető el. Már nem csak olvasni lehet róla. Már nem csak a méregdrága számítástechnikai rendszerek kiváltsága. Közelebb van mint gondolná! Képzelje el, hogy már az Ön cége számára is elérhető ez a szédületesen stabil szerver háttér! Képzelje el, hogy mindez kedvező áron megvásárolható! Képzelje el, hogy nem kell többé elképzelnie, mert a 99,999%-os megbízhatóság az Ön cége számára is valósággá válhat!

Ha a **Microsoft Windows 2000 Server** családot választja, akkor a megbízhatóság csúcsát választja.

Microsoft® Windows® 2000 Server Family

*Ez az adat függ az operációs rendszeren kívüli körülményektől is, mint például a hardvereszközök milyenségétől, illetve más, az operációs rendszertől független szoftverektől.

99999 A legendás öt kilences: 99,999%-os rendelkezésre állás.

http://www.microsoft.com/hun/products/windows.htm
http://www.microsoft.com/hun

Microsoft Ügyfélszolgálat: 2MSINFO [267-4636]
Microsoft Forródrót: 2MSSUGO [267-7846]

Microsoft

Hátrább az egerekkel!

Szerkesztőség
Főszerkesztő: **Fóti Marcell**
marcellf@netacademia.net
Főszerkesztő-helyettes: **Fülöp Miklós**
mick@netacademia.net
A szerkesztőség címe:
1062 Budapest, Andrássy út 62.
Tel.: 472-1214
technet@netacademia.net
Nyilvános levelezési lista:
tech.net@technetklub.hu

Kiadja és terjeszti a
NetAcademia Kft.
Terjesztési, előfizetési információ:
Tel.: 472-1214
terjesztes@netacademia.net
Megjelenik havonta, ára 1.344 Ft
Példányszám: 4.000

Előfizethető megrendelőlevélben a szerkesztőségnél:
1062 Budapest, Andrássy út 62.
Fax.: 472-1215
http://technet.netacademia.net/subs

Hirdetésfelvétel:**Szívós Éva**
Tel.: 472-1214
Fax.: 472-1215
info@netacademia.net
Grafiakai tervezés, kivitelezés:
Gregor László
Nyomdai előkészítés:
NetAcademia Kft.

Nyomda:
Hieron Kft.
2120 Dunakeszi, Tamás A. u. 11/a
Felelős vezető: **Török Andrea**

ISSN 1586-5185

Nemrégiben egyik kedves olvasónk kérdezte, mit gondolok a tech.net magazin gyakorlati hasznáról. Nyilván az a gondolat vezette e kérdésben, hogy ő maga nem találja a gyakorlati hasznot. Megkérdeztem, mit is vár tulajdonképpen. Kisült, hogy olyan jellegű click-by-click leírásokat hiányol, melyekből közel százezret tartalmaz a mindannyiunk által ismert tudásbázis *(Knowledge Base, http://support.microsoft.com)*, s amelyekre maximum hivatkozni szoktunk, de arra még soha nem volt példa, hogy egy az egyben beemeljük lapunkba. Ez - amíg én ülök a kormányrúdnál - nem is fog megtörténni, mert egy máshol már publikált, ráadásult porszáraz anyag lefordításának és közlésének nem látom semmi értelmét.

A gyakorlati haszonnál különben is előrébb való az „elméleti haszon", mert megfelelő elméleti felkészültség birtokában könnyebben megy minden munka: tervezés, hibaelhárítás, sőt, még a rendszerfelügyelet is!

A Microsoftnál végzett kutatások szerint *(egy régebbi, katasztrófaelhárítással foglalkozó MOC könyvben szerepelt a tortadiagram)* az informatikusok hibaelhárításban *(support)* mutatott sikerességét több, mint 60%-ban elméleti tudásuk alapozza meg, erre rakódik körülbelül 30%-nyi tapasztalat - és a fennmaradó tíz százalékon osztozik az intuíció, a szerencse és a tárgyi tudástól el nem vakított éleslátás *(a tudományban „vak tyúk effektus" néven ismert jelenség)*. Ezek a számok önmagukért beszélnek, de hogy konkrét példával is éljek, vegyünk mondjuk egy huncutkodó IPSec hálózatot, mely nyílt kulcsú technológiára épülő titkosítást használ. Vajon melyik supportőr fogja tudni hamarabb működőképes állapotba hozni a rendszert? Aki soha életében nem gondolta végig az IPSec működését, így „ösztönösen" választ néhány step-by-step, click-by-click, gyakorlati haszonnal kecsegtető leírást, és azokat szisztematikusan végigkattintgatja, vagy a másik, aki tudja, mettől meddig mennyi az IPSec?

Több tízezer Knowledge Base cikk birtokában hajlamosak vagyunk az alábbi hibaelhárítási sorrendet követni:

1. Újrabootolni. *(Ez mindig használ. Csakúgy, mint az orrszarvúkörömpor bizonyos pszichoszomatikus betegségek esetén.)*
2. KB cikkeket megnézni, néhányat kipróbálni. *(Ez is mindig beválik. Különösen a registry átírkálása...)*
3. Gondolkodni. *(Ennek működőképességéről nincsenek adataink.)*

Bevallom, időzavarban néha én is beleesek ebbe a csapdába, de azért élünk, hogy tévedjünk. Pár óra küszködés után azonban mindig eljutok a harmadik pontig, ahonnan általában könnyedén eljutok a megoldásig, ami néha csupán ennyi: az adott problémának egyáltalán nincs megoldása. Pont az ilyen esetek világítják meg legjobban az első két lépés értelmetlenségét: aminek nincs megoldása, annak nincs megoldása.

Hogy visszatérjünk a tudásbázishoz: 23348 darab olyan cikket tartalmaz, melyek „megoldhatatlan" problémáról szólnak. Hogy honnan tudom? Valahányszor egy problémára nincs valódi, szép, tudományos, frappáns vagy más pozitív jelzővel illethető megoldás, a Knowledge Base cikk egyszerűen WORKAROUND-nak nevezi a kiutat. *(Ha nem tudod átugrani, kerüld meg.)* Nos, rákerestem erre a kulcsszóra, és pontosan huszonháromezer-háromszáznegyvennyolc találatot kaptam.

Minden olyan esetben, amikor a problémának valódi megoldása van, a KB-cikk ezt tartalmazza: RESOLUTION. Ilyen leírásból 52956 darab volt 2002 májusában. Elképesztő számok, ugye? Sokan mondhatják, hogy csalok, mert ugyan kit érdekelnek a Word for Macintosh WORKAROUND-jai és a Lan Manager RESOLUTION-jai? Erre azt felelem: csak a Windows NT/2000 magánéletéről 2208 darab cikk szól!

Ezt a mennyiséget csak minőséggel lehet legyőzni. Tessék ismerni az informatikai rendszer elemeinek működési elvét! Mi ebben próbálunk meg segíteni - a Knowledge Base szűrögetése továbbra is mindenkinek önálló házi feladat!

Fóti Marcell
marcellf@netacademia.net
MZ/X

2002 augusztus - szeptember

A szoftver és én

Június 28-án, pénteken este hét órakor eléggé kétségbeesve hívtam fel egy jó barátomat és kollégámat, hogy a tanácsát kérjem. Rendes körülmények között egy jó barátot nem traktál az ember ilyenkor a munkahelyi problémáival, de ezúttal a körülmények nem voltak rendesek. Már délelőtt óta küzdöttem egy Exchange 5.5 szerverrel. A hibajelenség abból állt, hogy a kiszolgáló, mint az őrült *(ki letépte láncát…)*, százszámra ontott magából leveleket, tulajdonképpen kettőt-hármat újra meg újra, és továbbította azokat egy internetes cím felé. A szerencsétlen címzett több százszor kapta meg ugyanazokat az üzeneteket, én pedig semmit sem tudtam tenni. Ha bárhol megállítottam az áradatot, a levelek óriási sorokat képeztek, feltöltve a lemezeket. Úgy tűnt, hogy az üzenetek a „semmiből" jönnek, vagyis az Exchange Information Store komponensében „keletkeznek" és indulnak útjukra. A hibát záros határidőn belül kellett elhárítanom: másnap indult a repülőgépem Barcelonába és azt nem szerettem volna lekésni.

Mikor a kollégámnak elmeséltem a helyzetemet, és a fentieknél kicsit bővebben taglaltam a hibát, meg azt, hogy nincs a szituációt leíró KB cikk, és semmiféle egyéb segítségem sincs, megkérdezte tőlem: „Te még ezek után sem szidod ezt a szoftvert? Még most is lelkes vagy?"

Igencsak elgondolkodtatott a kérdés, olyannyira, hogy a hibát elfelejtve majd tíz percen keresztül beszéltem arról, mit is gondolok magamról, meg a szoftverről, meg a Microsoftról, meg a világról, ami ugyan bugos, tudjuk, de mégis…

Aztán befejeződött a telefonos segélyszolgálat, a kolléga pontos instrukciókat nem, de némi tanácsot és ötletet adott, amelyeket igyekeztem megszívlelni. A kérdése azonban nem hagyott nyugodni.

Sebtiben akkor azt találtam mondani a telefonba, hogy a viszonyom a rendszerrel olyan, mint a házasság. Ha a feleségem bal lábbal kel, attól még remek társ, és ha összeveszem vele, a házasságom tönkremehet, a bal-lábbal-kelés problémáját viszont egyáltalán nem oldom meg.

Az élet persze azt mutatja, hogy a bugos házastársakat egyáltalán nem viselik el az emberek, összevesznek velük, szidják őket, végül elválnak. A szoftverekkel is hasonló a helyzet: szidják, utálják őket, mert nem tökéletesek, és adandó alkalommal új verzióval, vagy épp teljesen mással próbálnak szerencsét. Hmm. Tökéletes férj és gyári hibás feleség? Hibátlan rendszergazda és alávaló szerver?

Egy riportfilm jutott az eszembe. A riporter csodálkozva nézte, hogy az asztalosfiú gyönyörű mintákat faragott a gyalujába. „Miért faragtad ki a gyaludat?" jött a naív kérdés, lebecsülő hanghordozással. *(Hiszen az csak egy gyalu…)* „Egész álló nap ezt nézem. Rondát lássak?" felelte a srác, s talán semmi sem fejezhette volna ki jobban a munkája iránti szeretetet, mint éppen ez a kérdés: „Rondát lássak?" Ez az ember nem fogja eldobni a gyaluját, talán egész életében…

A klasszikus hozzáállás az, hogy minden és mindenki hibás, felelős és bugos: a szoftver, amelyet egy rossz cég rosszul írt meg, felületesen ellenőrzött és felelőtlenül adott ki; az IT vezetés, amely oktondi módon bedőlt a reklámoknak és megvette a gagyi terméket; a szállító, amely rosszul telepítette, a támogató szervezet, amely nem támogat; minden és mindenki tökkelütött, csak ÉN, a tökéletes, fantasztikus, zseniális műszaki szakember, csak én nem vagyok hibás. Én mindent tudok: az összefüggések a kisujjamban, elméleti tudásom alaposságánál csak az évtizedes *(sőt, mondjuk ki: évszázados)* tapasztalatom híresebb, szóval csak én nem… Hmm.

Egy ősi kínai *(vagy hindu, vagy perzsa, vagy héber, vagy egyiptomi vagy görög)* bölcs azt mondta, a bölcsesség önnön oktondiságunk és tökéletlenségünk felismerésével kezdődik. Vagyis: a fa megfelelő, a gyalu éles, csak én vagyok ügyetlen, hogy megmunkáljam. A feleségem nagyszerű társ, csak én ezt hajlamos vagyok elfelejteni. A szoftver jó, de nem ismerem eléggé. A munka nagyszerű, csak bennem nincs kellő alázat iránta…

Alázat, érdekes szó ugye? Hallotta már a Kedves Olvasó ezt a szót informatikával foglalkozó szakemberrel kapcsolatban emlegetni? Azt mondják, hogy az alázat nem a fiatalság erénye. Az is tény, hogy a kollégák többsége fiatal. Épp ezért talán az a baj, hogy nincs „mesterük", aki megtaníthatná nekik, hogy ne akarjanak tökéletes szakemberekké válni. Legyenek tökéletes emberek, váljanak azzá a munkájuk révén - a hozzáértés csak a hozadék, az alázat jutalma.

Szóval kezdjük az elején: nem tudok semmit, nem értek semmit, de szeretnék tanulni. A hiba egy lehetőség, hogy tanuljak, megtanuljam, hogy mi miért úgy működik, ahogy. A körülöttem lévőket nem minősítem, hiszen úgyis csak az a fontos, ők hogyan minősítenek engem. Nem méregetem a munka tárgyát *(Exchange)*, hiszen úgyis az tesz mérlegre engem, meg tudom-e javítani vagy sem. Megértem-e vagy sem?

És íme még egy titok: ha tökéletesedem, akkor a körülöttem lévő dolgok is tökéletesednek. Ez kisugárzik. Nem éppen tudományos módon sugárzik, de mégis… Vagyis egy idő után a szoftverek valamiképp úgy működnek, ahogy kell. És mindez visszafelé is igaz: ha nem szeretem azt, amivel dolgozom, az bosszút áll rajtam. Nálam nem fog működni. A dolgok már csak ilyenek…

Az idő fogyott, és az utolsó mentés visszaállítása sem segített: a levelek záporoztak. Valószínűleg nem zárta le a „küldés" tranzakciót az Information Store, és újra meg újra megismételte, aztán újra nem tudta lezárni a végtelenségig… Nem mintha ez bárhol olvasható lenne, de eléggé logikusnak tűnő magyarázat. Azt találtam gondolni, hogy feltételt szabok az MTA-n való áthaladáshoz: nem mehet át az a levél, amely nagyobb, mint 1K. A forgalom megállt, a sorok kiürültek. A korlátot töröltem. Másnap repültem Barcelonába.

Lepenye Tamás, MCSE 2000

Farkasokkal táncoló

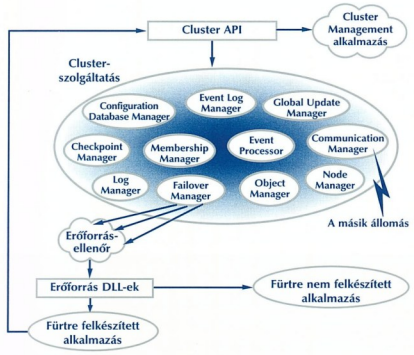
(X. rész) - Cluster a gyakorlatban

Amikor a hibakeresést ismertettem, futólag említettem a cluster.log állományt is. Akkor nem tudtam kellő figyelmet szentelni ennek az eszköznek, csupán néhány fontos tudnivalót írtam le. Most újabb alapozásba fogunk, elmélyedünk a fürt belső világában és szerkezetében, hogy azután később megismerhessük a fürt speciális eseménynaplóját is.

Egy állomás belső architektúrája

A cluster.log elemzését csak akkor lehet elkezdeni, ha tisztában vagyunk a fürt és állomásainak belső architektúrájával. Mindeddig nem beszéltünk erről, mert nem volt érdekes. Most azonban nem kerülhetjük meg, mert különben a cluster.log érthetetlen zagyvaság marad a számunkra.

A cluster egyik állomásának fürtszolgáltatását *(cluster service)* vázolja az 1. ábra



♣ *A clusterszolgáltatás architektúrája*

Vegyük sorra az egyes komponenseket:

A cluster management alkalmazás nem más, mint a fürtadminisztrátor *(cluadmin.exe)*. Ez a program a speciális fürtalkalmazás API-n keresztül tartja a kapcsolatot a clusterszolgáltatással *(cluster.exe)*.

A fürt egy állomáson három fő alkotóelemből áll: a fürtszolgáltatásból, az erőforrás-ellenőrből *(Resource Monitor)* és az erőforráskönyvtárakból.

A cluster.exe több belső komponensből épül fel, amelyek együttesen valamennyi fürt-specifikus tevékenységet ellátják. Már tudjuk, hogy minden állomáson fut egy-egy fürtszolgáltatás, de a belső komponensekről és azok funkcióiról még nem volt szó. Vegyük őket sorra:

- **Checkpoint Manager [CP]** - A komponens feladata az alkalmazások regisztrációs kulcsainak mentése a quorum lemez MSCS könyvtárába. Ez teszi lehetővé, hogy egy fürtre fel nem készített alkalmazás átköltözés esetén újra tudjon indulni. A CP ellenőrzi a regisztrációs kulcsokat az erőforrás indulásakor és ellenőrzőpontot *(checkpont)* ír a quorumba, amikor az erőforrás leáll. A fürtre felkészített alkalmazások a konfigurációs adatbázist, a fürtre fel nem készítettek pedig a helyi szerver regisztrációs adatbázisát használják a helyreállításhoz szükséges információk tárolására. Ezt az információt „vezeti át" a quorumba a CP, ezért csak a fürtre fel nem készített alkalmazásoknál van rá szükség.
- **Log Manager [LM]** - A chekpoint managerrel együtt azt biztosítja, hogy a quorumerőforrásban a legfrissebb, helyreállításhoz szükséges adatokat tartalmazza. Nem szabad összekeverni az esemény-feldolgozó [EP] szállal. *(Lásd később)*
- **Communications Manager [CIMsg]** - A fürtállomások közötti kommunikációért felelős programkód. Az RPC-t használó komponens biztosítja, hogy minden fürtön belüli üzenet eljusson a többi állomásra méghozzá pontosan egy alkalommal. Ha egy üzenet egy olyan állomásról érkezik, amely a konfigurációs adatbázis szerint már nem tagja a fürtnek, a szolgáltatás az üzenetet eldobja.
- **Configuration Database Manager [DM]** - Ez a fürtkonfiguráció, vagyis egy speciális adatbázist kezelő szál. Ez az adatbázis tárolja a fürt fizikai és logikai entitásait. Ilyen entitás maga a fürt, a fürttagság, az erőforráscsoportok és erőforrások *(például az IP-címek vagy a fizikai lemezek)*. Az állandó és ideiglenes adatok, amelyek az adatbázisban találhatók, arra szolgálnak, hogy érzékelni lehessen a fürt állapotát és állapotváltozásait. Minden állomás DM-e együttműködik más állomások hasonló komponenseivel, hogy a konfigurációs információk megegyezzenek valamennyi node-on. A DM ezen kívül még egy registryhez hasonló felületet is nyújt a többi clusterkomponens számára. Ezen a felületen érkeznek be a változtatási kérelmek az adatbázisba.
- **Global Update Manager [GUM]** - Ezt a komponenst a Configuration Database Manager használja arra, hogy minden konfigurációs változás valamennyi állomáson érvényre jusson. A GUM biztosítja, hogy a változásokról minden állomás értesüljön. Ha valamelyik állomás nem végzi el a frissítéseket, akkor „el kell hagynia" a fürtöt, mert az különben nem tudna konzisztens állapotba kerülni. A leggyakrabban úgy találkozunk majd a GUM

A cluster.log elemzését csak akkor lehet elkezdeni, ha tisztában vagyunk a fürt és állomásainak belső architektúrájával.

bejegyzéssel, hogy a DM utasítására egy meghatározott lépésekből álló tranzakciót végez a quorumon. Lefoglalja az adatbázist, elvégzi a módosítást, majd feloldja a foglalást.

- **Event Processor [EP]** - Ez a részegység fogadja az eseményeket a fürterőforrásoktól. Egyfajta elektronikus kapcsolótábla, amely a fürtszolgáltatás és a fürtözött alkalmazások között küldözgeti az eseményeket. Az eseményeket minden olyan komponens megkapja, amelyik támogatja a fürt API eseményfogadás mechanizmusát.
- **Event Log Manager [ELM]** - A fürtszolgáltatás e szál segítségével másolja oda-vissza az eseménynapló bejegyzéseit az állomások között. Itt nem a cluster.log-ról, hanem a Windows 2000 saját eseménynaplójáról van szó. Ez jelentősen megkönnyíti a hibakeresést, mert akkor is láthatjuk a másik állomás eseményeit *(vagy legalább egy részüket)*, ha a node egyáltalán nem elérhető.
- **Failover Manager [FM]** - A részegység feladata az erőforrások kezelése, valamint különböző tevékenységek indítása, például egy csoport átköltöztetése vagy újraindítása. A függőségek kezelése is itt történik. Az FM folyamatosan állapotinformációkat kap az erőforrás-ellenőröktől és az állomásoktól. Ez a komponens dönti el, hogy melyik erőforráscsoportot melyik állomás birtokolja. Amikor az erőforráscsoportok kezdeti leosztása *(arbitration)* megtörtént, az egyes állomások „birtokba veszik" az erőforrásokat. Ha egy erőforrás meghibásodik egy csoporton belül, és a szabályok szerint a meghibásodást egy állomás nem tudja lekezelni, az FM menedzserek mindkét állomáson együttesen újraosztják az erőforráscsoportokat.
- **Membership Manager [MM] [RGP]** - Ellenőrzi a fürttagokat, és figyeli, hogy más állomások megfelelően működnek-e. Leginkább a node-ok indulásakor, leállásakor vagy rendellenes „eltűnésekor" találkozhatunk vele.
- **Node Manager [NM]** - Mindegyik állomáson fut, és helyi listát vezet arról, hogy mely állomások tartoznak a fürthöz. A Node Manager rendszeres időközönként szívhang-üzenetet küld a másik állomásnak, hogy adott esetben felfedezze annak meghibásodását. Alapvető, hogy minden állomás egy fürtön belül azonos módon lássa a fürttagságra vonatkozó információkat. Ha egy állomás kommunikációs hibát észlel egy másik állomással, szórt üzenetet küld minden fürttagnak, hogy ellenőrizzék a fürttagsági információkat. *(A folyamatnak két állomás esetén nincs sok értelme, de három vagy négy állomásnál már fontos lehet.)* Ezt az ellenőrzést újracsoportosításnak *(regroup event)* hívják. A fürtszolgáltatás mindaddig tiltja a közös lemezekre való írást, amíg a tagság nem tisztázódik. Ha egy Node Manager egy állomáson nem válaszol, a többiek eltávolítják a fürttagságát, és az aktív erőforráscsoportjait átmozgatják a megfelelő állomásokra attól függően, hogy melyek a csoport preferált és lehetséges állomásai. Két állomás esetén ez megint egyszerű, három- illetve négy node-nál viszont eléggé bonyolult erőforráscsoport-elosztás is lehetséges.

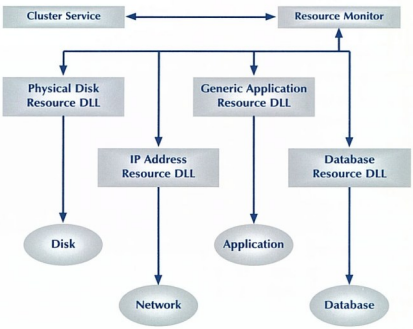
A Node Manager rendszeres időközönként szívhang-üzenetet küld a másik állomásnak, hogy adott esetben felfedezze annak meghibásodását.

- **Object Manager [OM]** - Ahogy a neve is mutatja, ez a szál a fürt objektumait kezeli, különböző *(belső)* objektumok létrehozását, keresését, kiértékelését végzi el.

Még egyszer tehát: a fenti objektumok mind a fürtszolgáltatás részei. A nevek mellett feltüntettem azokat a rövidítéseket is, ahogyan az egyes komponensekre a cluster.log hivatkozni fog. Az első ábrát tovább szemlélve láthatjuk, hogy a Failover Manageren keresztül tartja egymással a kapcsolatot a cluster.exe és az erőforrásellenőr, amely a második építőkocka a fürtünkben.

Az erőforrásellenőr *(Resource Monitor)* [RM]

Ez a komponens egy különálló processz, amely egységes felületet nyújt a fürtszolgáltatás és az erőforrások összekapcsolásához. A clusterszolgáltatás valójában az erőforráskönyvtárakkal *(DLL-ekkel)* kommunikál. Az erőforrásellenőr tehát egyfajta pajzs is, amely megvédi a fürtszolgáltatást egy hibásan működő erőforrástól.



♣ A fürtszolgáltatást több réteggel elválasztották a tényleges erőforrásoktól

Az ellenőr több példányban is futhat. Ez azért fontos, mert ha van egy bizonytalanul működő erőforrás, az elszigetelhető a többitől egy saját erőforrásellenőrrel. Saját ellenőrt úgy rendelhetünk az erőforráshoz, ha a létrehozásakor vagy a tulajdonságai lapján bejelöljük a „Run the resource in a separate Resource Monitor" választónégyzetet. Az ellenőr a fürtszolgáltatással a kapcsolatot RPC hívásokon keresztül tartja.

Az erőforráskönyvtárak *(Resource DLL)*

A fürtöt alkotó harmadik építőelem-csoport az erőforráskönyvtáraké. A Windows 2000 Advanced Server önmagában is számos ilyen könyvtárral rendelkezik a különböző típusú erőforrásokhoz, ezek köre az alkalmazások telepítésével kibővül. Ha egy alkalmazásnak van saját erőforráskönyvtára, továbbá a fürtszolgáltatással a fürt API-n keresztül kommunikál, akkor azt mondhatjuk, hogy az egy fürtre felkészített *(cluster-aware)* alkalmazás.

A saját erőforráskönyvtárral nem rendelkező alkalmazásokat a fürt ún. általános alkalmazásokként *(generic application)* vagy általános szolgáltatásokként *(generic service)* kezeli. Az ehhez tartozó könyvtárállomány *(clusres.dll)* a fürt beépített erőforrásállományai.

A fürtre felkészített erőforrásokat természetesen jobban kezeli a cluster. Egy ilyen alkalmazás például:

- képes leírni a saját állapotát az erőforrásellenőr kérésére,
- teljesen szabályosan, az adatintegritásra ügyelve képes leállni és újraindulni, valamint
- precízebben válaszol a „valóban él" *(IsAlive)* és az „úgy tűnik él" *(LooksAlive)* vizsgálatokra.

Egy erőforráskönyvtár több erőforrástípushoz is tartozhat. Ha egy ilyen DLL megsérül, mindazon erőforrások nem indulnak majd, amelyek használják a könyvtár függvényeit, és persze a velük függőségi viszonyban lévők sem.

Az első ábrát majdnem teljes egészében értelmeztük. Csupán egyetlen kapcsolatot, a fürtre felkészített alkalmazások és a cluster API közötti összefüggést kell tisztázni. Ez a kapcsolat úgy jön létre, hogy a fürtre kész alkalmazások a fürtadminisztrátort kiegészítő állományokkal látják el - ezzel lehetővé téve, hogy az erőforrás egyedi paramétereit a kezelőfelületen keresztül állíthassuk. Amikor egy Exchange 2000 erőforrás egyedi paramétereinek „fülére" kattintunk, tulajdonképpen ezeket a kiegészítő állományokat használjuk.

Az állomások együttműködése

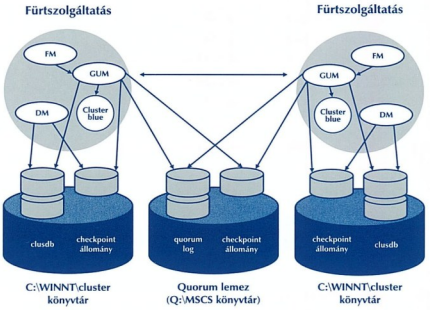
Egy állomás nem állomás. Nézzük meg, hogyan dolgozik együtt két Windows 2000 Server.

A fürtszolgáltatás egyik nagy problémája, hogy pontos információval kell rendelkeznie azokról a más állomásokon található fürtszolgáltatásokról, amelyek társak lehetnek az erőforrások üzemeltetésében. Ha bármilyen állapotváltozás történik, egy megfelelő mechanizmusnak gondoskodnia kell arról, hogy a teljes fürt érzékelje az állapotváltozást. A Windows 2000 Server ehhez több példányban és több helyen adatbázisokat tárol, frissít és szinkronizál.

A fürtszolgáltatás telepítésekor egy közös lemezen, a quorumon létrejön egy quorum.log nevű állomány. Szerkezetét tekintve ez regisztrációs adatbázis ág *(registry hive)* - elegendő, ha mi most „adatbázis"-ként definiáljuk.

Ezek az adatok azonban nem csak itt érhetők el. Mindegyik állomás tartalmaz egy-egy saját példányt is a C:\WINNT\cluster mappában clusdb néven. A szolgáltatás indulásakor ez a fájl kerül először a memóriába, ebből lesz a cluster ág a regisztrációs adatbázisban. Két állomás esetén összesen öt példányban létezik az a „tudás", amit a node-oknak birtokolniuk kell. Ezek az adatbázispéldányok nem tökéletesen egyformák *(nem konzisztensek)*, de ha mindkét állomáson szabályosan leállítanánk a clusterszolgáltatást, a lemezeken tárolt példányok azonos állapotba kerülnének. *(Ha tehát a quorum.log megsérül, nem nehéz kitalálni, honnan pótolhatjuk.)*

A sok adatbázispéldány ugyanakkor megnehezíti a változások átvezetését. A későbbi logelemezésnél láthatjuk majd, hogy a legtöbbet a Global Update Manager *(GUM)*, a Database Manager *(DM)* és a Failover Manager *(FM)* kezdeményez frissítési műveletet. Az alábbi ábra ezeket a komponenseket mutatja. Természetesen a többi belső komponens is ide tartozik, de azok megjelenítése zavarólag hatott volna. Ezt a képet fejben tartva jobban megérthető a fürtök belső működése.



♣ ***Két állomás együttműködése***

A működési folyamat leegyszerűsítve a következő:

Az FM az erőforrásellenőr segítségével folyamatosan figyeli a rábízott erőforrások állapotát. Ha bármilyen okból állapotváltozás történik, a GUM komponens segítségével értesíti a „túloldali" FM-et, és frissíti a memóriában található adatbázispéldányt. A DM szintén a GUM segítségét veszi igénybe, amikor a helyi adatbázist *(clusdb)*, vagy a közös adatbázist *(quorum.log)* írja. A GUM egyik legfontosabb feladata a különböző adatbázispéldányok zárolása az írási műveletek számára.

A cluster.log alapjai

Az alapvető tudást megszereztük a fürt saját eseménynaplójának értelmezéséhez. Most nézzük meg, hogyan néz ki egy be jegyzés az állományban.

```
00000dd8.00000c6c::2002/06/23-11:37:56.089 [CS]
Cluster Service started - Cluster Node Version
3.2195

00000dd8.00000c6c::2002/06/23-11:37:56.089
OS Version 5.0.2195 - Service Pack 2 (AS)
```

A naplóbejegyzés mindig annak a processznek és szálnak az azonosításával kezdődik, amelyek a bejegyzést végezték. A két azonosítót egy pont *(.)* választja el egymástól. A mi esetünkben a processz ID dd8, míg a szálazonosító c6c.

Ezután következik az időpont bejegyzése, méghozzá minden időzónában a greenwichi idő szerint. Ez nálunk két óra időeltolódást jelent, elsőre ijesztő, hogy „rosszul jár" a cluster.log órája. Az időbélyeg formátuma éééé/hh/nn-óó:pp:mp:emp, vagyis ezredmásodpercre pontos bejegyzés készül. Végezetül jön az esemény leírása - itt épp elindul a fürtszolgáltatás. A cluster.log-ban tulajdonképpen kétféle bejegyzés található: a fürtszolgáltatás komponenseitől és az erőforráskönyvtáraktól származó. Ha egy komponens rögzít egy eseményt, egyúttal elhelyezi a saját rövidítését is - ezeket tüntettem fel a szögletes zárójelek között a nevek után. A felsoroltakon kívül azonban még van néhány rövidítés, amellyel meg kell ismerkednünk:

A cluster.log-ban tulajdonképpen kétféle bejegyzés található: a fürtszolgáltatás komponenseitől és az erőforráskönyvtáraktól származó.

- API - A fürt API felületet nyújtó komponenstől érkező bejegyzések rövidítése.
- CINet - A fürt hálózati motorja.
- CS - A fürtszolgáltatás. A bejegyzés egy része nem a komponensekhez kötődik, ekkor kapja az általános CS jelzést. A fenti bejegyzés jó példa, a fürtszolgáltatás indulásakor keletkezett.
- INIT - Ez nem egy komponens, hanem egy státusz jelzése. Akkor kapja ezt egy bejegyzés, ha clusterszolgáltatás még nem csatlakozott egy másikhoz, vagy önmaga nem alakított ki egy saját fürtöt.
- JOIN - Az INIT állapot utáni státusz. Ha a kapcsolódás sikeres, az állomás fürttag lesz.
- EVT - Nem dokumentált a pontos jelentése, az „esemény" szó rövidítése.
- CPROXY - A rövidítés annyit tesz: cluster proxy, vagyis fürthelyettesítő. Egy nagyon kicsi alkalmazásról van szó, amely a Windows NT 4.0 egyik fogyatékosságát, a szervízellenőr *(Service Control Manager)* hiányát segít orvosolni. A fürt indításakor technikai értelemben a CPROXY-t indítjuk el, amely rögtön betölti a valódi fürtszolgáltatást. Emellett érzékeli, ha a valódi cluster összeomlik, és megpróbálja újraindítani. A Windows 2000-ben a CPROXY trükkre már nincs szükség, a fejlesztők azonban nem végeztek teljes átalakítást a fürtszolgáltatás szerkezetében, ezért a módszer maradt. A következő verziókban vélhetően már nem találkozunk vele.

A rövidítésekhez további két apróságot kell ismerni. A Membership Manager [MM] néha [RGP] néven is megjelenik - ezért mindkét rövidítést szerepeltettem a leírásánál.

Az INIT és a JOIN rövidítésekről azt kell tudni, hogy hozzákapcsolódhat más rövidítésekhez is, egyszerre jelezve egy komponenst és egy állapotot. Például NMJOIN.

A komponensek bejegyzései mellett az erőforráskönyvtárak írnak az eseménynaplóba. Egy lemezerőforrás-bejegyzés például így néz ki:

```
15c.458::1999/06/09-18:00:47.897 Physical Disk
<Disk D:>: [DISKARB] Arbitration Parameters (1
9999).
```

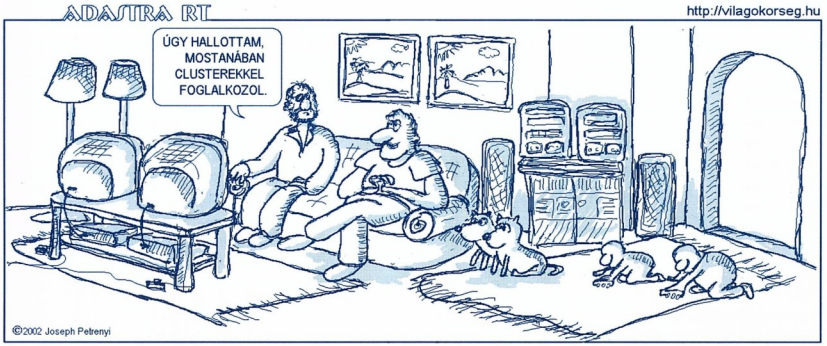
Az alapvető különbség a két bejegyzés között az, hogy az erőforráskönyvtárak nem helyeznek el rövidítéseket, legalábbis nem az időbélyeg után. A napló olvasásánál ezért jól el lehet különíteni, hogy honnan származik az esemény.

> Egy csoport átköltöztetése csupán hat másodpercig tart, a fürt ezalatt 3500 (!!) eseményt talál méltónak arra, hogy a saját naplójába bejegyezzen.

A cluster.log olvasása nem azért nehéz, mert bonyolult a szintaxisa. Igaz, talán lehetne egy kissé dokumentáltabb az eszköz - némi gyakorlás után azonban ez nem jelent gondot. A probléma abból fakad, hogy a bejegyzések száma óriási. Egy csoport átköltöztetése csupán hat másodpercig tart, a fürt ezalatt 3500 (!!) eseményt talál méltónak arra, hogy a saját naplójába bejegyezzen. És ez csak az érem *(vagyis a fürt)* egyik oldala, mert az eseményekről a másik állomás is értesül, és azokat aztán ő is szorgosan jegyzeteli.

Azt gondolom, hogy az elemzés elkezdését a következő alkalomra hagyjuk. Lesz bőven tenni- és megértenivalónk.

Lepenye Tamás, MCSE 2000
lepenyet@mal.hu



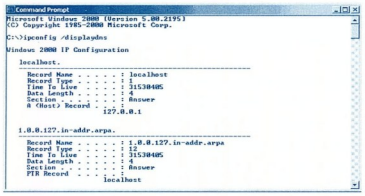
DNS és névfeloldás a Windows 2000-ben (2. rész)

Az előző részben belekezdtünk a Windows 2000 névfeloldási műveletének boncolgatásába. Eljutottunk addig, hogy a DNS-ügyfél (azaz a kliensgép) kérdésére választ kap: ez a válasz lehet pozitív, a kérdéses IP-címmel; és lehet negatív is, ha a cím nem létezik. Lássuk, mi történik a válasszal...

Az ügyféloldali DNS-gyorsítótár

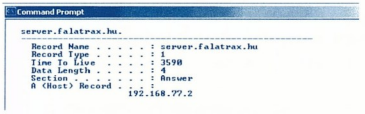
Minden DNS-lekérdezés eredménye, legyen az pozitív vagy épp negatív, bekerül a Windows gyorsítótárába. Ezután mielőtt bármely beállított DNS-kiszolgálóval felvenné a kapcsolatot, belenéz a saját tárba, és ha ott megtalálja a kérdéses információt, fel is használja azt. Mielőtt azonban a gyorsítótár működését részletesen elemeznénk, lássuk, hogyan tekinthetjük meg a tartalmát! A varázsszó a következő:

```
ipconfig /displaydns
```



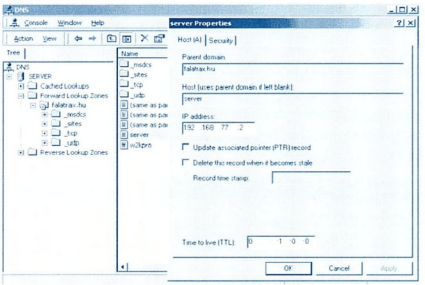
♣ A helyi DNS-gyorsítótárba az ipconfig /displaydns parancs segítségével leshetünk bele

Ez a lista kezdetben a helyi bejegyzéseket *(localhost, illetve a 127.0.0.1 címhez tartozó reverse rekordot)*, valamint a helyi HOSTS fájlból kiolvasott információkat tartalmazza. A lista ezután persze folyamatosan bővül, lássuk például, mi történik, miután kiadtuk a ping server.falatrax.hu parancsot *(tegyük fel, hogy ez a bejegyzés létezik a DNS-kiszolgáló zónájában)*!



♣ A visszafejtett címek bekerülnek a DNS-gyorsítótárba

Ha ismét kilistázzuk a gyorsítótár tartalmát, láthatjuk, hogy a lista bővült: ezúttal - mint az ábrán is látható - már a server.falatrax.hu cím is megtalálható benne. Figyeljük meg a Time To Live sorban található értéket! Az itt látható szám másodpercben jelzi azt az időt, amíg ez a bejegyzés érvényesnek tekintendő. Ha ezalatt az idő alatt újra szükségünk lenne a fenti IP-címre, azt a Windows automatikusan a gyorsítótárból olvasná ki. Azt, hogy egy adott bejegyzést az ügyfél mennyi ideig tárolhat saját gyorsítótárában, a kérdéses DNS-rekord TTL-tulajdonsága határozza meg. Ellenőrzésképpen nyissuk meg a DNS-kiszolgálón a server.falatrax.hu rekord tulajdonságlapját!



♣ A TTL-érték a DNS-rekord tulajdonságlapjának aljáról olvasható le

Vigyázat, ez a sor csak akkor látszik, ha a konzolban a View menüben bekapcsoltuk az Advanced opciót!

Lássuk csak: a TTL-érték formátuma nap:óra:perc:másodperc, azaz esetünkben 1 óra. 1 óra = 60 perc = 3600 másodperc; helyben vagyunk tehát.

Nézzük csak meg az első ábrán látható TTL-értéket! A számítógép saját bejegyzéseinek TTL-értéke az ábrán 31530405 másodperc, amiről rövid fejszámolás után kiderül, hogy kicsit kevesebb, mint 365 napot jelent. Az automatikus, saját bejegyzések tehát egy évig vannak érvényben a helyi DNS-gyorsítótárban. Hogy azután mi történik, nem próbáltuk ki, de reméljük, az adatok érvényessége további egy évnyi időtartamra meghosszabbodik.

> Minden DNS-lekérdezés eredménye, legyen az pozitív vagy épp negatív, bekerül a Windows gyorsítótárába.

A gyorsítótár törlése

Ha a rekord TTL-időtartama alatt az adat a kiszolgálón megváltozik, mi értelemszerűen nem fogunk róla tudomást szerezni, hiszen a számítógépünknek eszébe sem jut, hogy a DNS-kiszolgálóhoz forduljon a kérésével. A gyorsítótár tartalmát viszont igény szerint bármikor törölhetjük is, az

```
ipconfig /flushdns
```

paranccsal. A tár ilyenkor alapállapotba kerül, azaz csak a saját bejegyzések lesznek benne megtalálhatók.

A negatív gyorsítótár

Mi a helyzet akkor, ha egy kérdéses cím nem létezik? Tegyük fel, hogy megpróbáljuk lekérdezni a teszt.falatrax.hu címet, mondjuk egy ping parancs közvetett segítségével:

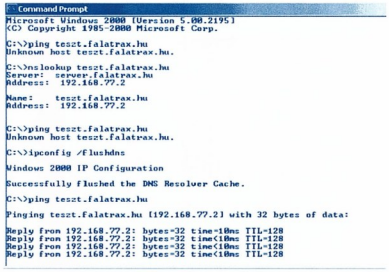
```
C:\>ping teszt.falatrax.hu
Unknown host teszt.falatrax.hu.

C:\>_
```

A cím a kiszogáló DNS-zónájában nem létezik, ezért egy negatív választ küld az ügyfélnek. Ha Network Monitorral figyelnénk a hálózatot, azt látnánk *(illetve nem látnánk)*, hogy a további próbálkozások alkalmával a Windows már nem próbálkozik meg a cím ismételt visszafejtésével. Ez pedig azt jelenti, hogy valahol az elutasító válasz is tárolódik. Nem kell sokat gondolkodnunk, hogy vajon hol is: lessünk bele ismét a helyi DNS-gyorsítótárba!

♣ ***A helyi gyorsítótár nemleges válaszokat, úgynevezett negatív bejegyzéseket is tartalmazhat***

A negatív gyorsítótár érdekes dolgokat művelhet, ha nem figyelünk oda. A következő példában ezt mutatjuk be: tegyük fel, hogy megpróbáljuk elérni a *(DNS-ben még nem létező)* teszt.falatrax.hu gépet!



♣ ***A negatív gyorsítótár szórakozik velünk :-)***

Az első ping parancsra kapott válasz így érthetően „Unknown host teszt.falatrax.hu". Tegyük fel továbbá, hogy a parancs kiadását követően a rendszergazda a DNS-ben létrehozta a szükséges bejegyzést *(így tettünk mi is)*. Az ábrán is látható, hogy az nslookup parancs segítségével lekérdeztük a teszt.falatrax.hu névhez tartozó címet, és azt vissza is kaptuk *(192.168.77.2)*. Rögtön ezután ismét megpróbáljuk megpingelni a gépet, és… a válasz megint „Unknown host teszt.falatrax.hu". Itt valami turpisság lesz: most létezik a cím vagy sem? Nyilván létezik, hiszen létrehoztuk, és az nslookup parancsra is megkapjuk a választ. Ha viszont ezen a ponton belenéznénk a gyorsítótárba, továbbra is ott találnánk a negatív bejegyzést! A dolog magyarázata az, hogy az nslookup parancs nem használja a helyi DNS-gyorsítótárat, se pro, se kontra: nem néz bele a címek lekérdezése előtt, és a választ sem helyezi el ott. Az nslookup feladata egyedül az, hogy a DNS-kiszolgálóval kommunikáljon, működése független a gyorsítótártól. Küldünk egy kérést, érkezik egy válasz, de mindennek semmi köze a cache-hez. Az tisztán látszik, hogy a kiszolgáló már válaszolna, ha a Windows megkérdezné, tehát nem maradt más hátra, mint a gyorsítótár törlése. És lássunk csodát, az ipconfig /flushdns parancs kiadása után a ping is azonnal működni kezd!

A negatív bejegyzések élettartama

Talán az Olvasónak is feltűnt, hogy a negatív bejegyzés alatt nem látható a TTL-érték bejegyzése. A negatív bejegyzéseket a Windows 2000 egy, a registryben beállított alapértelmezett ideig tárolja. A gyorsítótár beállításait a

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
Services\Dnscache\Parameters
```

kulcs alatt találjuk.

A negatív bejegyzések tárolásának alapértelmezett időtartamát a NegativeCacheTime érték tartalmazza, ez gyárilag 300 másodperc. Ugyanitt láthatjuk többek között még a MaxCacheEntryTtlLimit értéket is. Bár a pozitív válaszokat a DNS-kiszolgálótól érkező adatok alapján tároljuk, de ez az érték nem haladhatja meg az itt beállított időtartamot *(ami alapértelmezésben 86400 mp, azaz egy nap)*.

Jegyezzük meg, hogy negatív gyorsítótár-bejegyzés csak akkor jön létre, ha a DNS-kiszolgáló negatív választ ad egy kérésre. Ha a DNS-kiszolgáló nem érhető el, nem készül negatív bejegyzés.

A DNS-lekérdezések időzítése

Az előző rész végén már említettük, hogy a NetBIOS-névfeloldást a Windows 2000 csak akkor kísérli meg, ha a DNS-névfeloldás végképp csődöt mondott. A „végképp" szó pontosabb meghatározásához tegyünk ismét egy próbát, próbáljuk meg visszafejteni a www.blabla.hu címet, és közben figyeljük a hálózati forgalmat! *(Az ügyfélszámítógépen egy DNS-kiszolgáló címét adtuk meg.)*

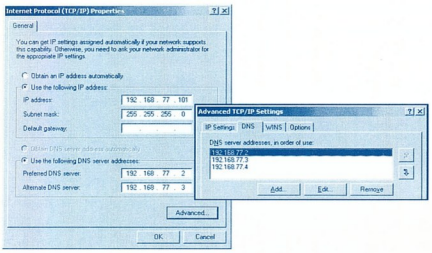
A negatív gyorsítótár érdekes dolgokat művelhet, ha nem figyelünk oda.

♣ ***A sikertelen DNS-visszafejtés után következhet csak a NetBIOS címfeloldás***

Nézzük először a DNS-lekérdezéseket. A Windows összesen öt alkalommal küld DNS-kérést a kiszolgálónak *(több kiszolgáló esetén ez kicsit másképp alakul, de arról majd később)*. Figyeljük a Time oszlopot! Az első kérést a következő kb. 1, azután 2, majd ismét 2, végül 4 másodperc múlva követi. Ha az utolsó kérésre sem érkezik válasz 8 *(itt, nemhivatalos mérés szerint kb. 7)* másodperc múlva, a Windows belekezd a NetBIOS-névfeloldásba. A DNS működésképtelensége tehát 1+2+2+4+8, azaz tizenhét másodperc „várakozást" jelent. A NetBIOS-kérést a Windows még kétszer ismétli meg, kb. egy-egy másodperc várakozás után, ami azt jelenti, hogy mire kiderül, hogy nincs ilyen cím, nagyjából húsz másodperc telik el.

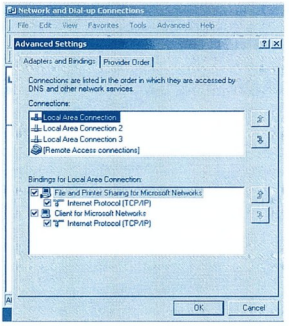
A DNS-kiszolgálók címének beállítása

Míg Windows NT 4.0-ban a DNS-kiszolgálókat a hálózati beállítások között „globálisan" adhattuk meg *(azaz a DNS-kiszolgálókat a Windows NT minden hálózati kártyán - alhálózaton - keresztül egyidejűleg kereste)*, a Windows 2000 ebben a tekintetben is okosabb; a DNS-kiszolgálók címeit minden hálózati csatolón külön-külön állíthatjuk be, és a Windows az egyes kiszolgálókat csak azon az alhálózaton keresi, ahol azokat definiáltuk.



♣ ***A DNS-kiszolgálók IP-címeit kártyánként adhatjuk meg***

A hálózati kártyán található TCP/IP tulajdonságok párbeszédablak első oldalán azonban csak az elsődleges és másodlagos kiszolgáló IP-címét írhatjuk be. Ha többet is meg szeretnénk adni, az Advanced gombra kattintsunk, és az így megjelenő dialógusablak DNS oldalán vegyük fel szépen sorban a címeket. Itt a kiszolgálók sorrendjét is könnyebben módosíthatjuk. Meg kell ismernünk még az „elsődleges hálózati csatoló" fogalmát is - mert bizony ilyen is létezik. Hiába van a Windowsban egynél több hálózati kártya, az egyenlők között van egyenlőbb, ami többek között meghatározza azt, hogy a Windows mely csatolón keresztül próbálja meg elérni a tartományvezérlőjét; letölteni a csoportos házirendeket; vagy éppen - és innentől kezdve fontos a dolog számunkra is - milyen sorrendben kezdi el lekérdezni a DNS-kiszolgálókat. Kártyánként ugyanis meghatároztunk egy sorrendet, és ez rendben is van - de mi legyen a helyzet a kártyák közötti sorrenddel?



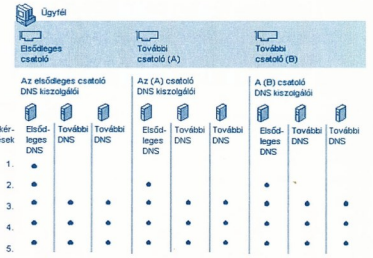
♣ ***Ha több hálózati kártyánk van, számít, hogy milyen sorrendet állítunk fel közöttük***

A csatolók sorrendjéhez a My Network Places Advanced menüjének Advanced Settings... menüpontjára kattintva megjelenő dialógusablakban juthatunk hozzá. Amint az ábrán is látható, a lista melletti gombok segítségével a sorrendet módosíthatjuk. Amelyik csatoló itt a lista tetején áll, azt nevezzük „elsődleges hálózati csatolónak".

A DNS-kiszolgálók lekérdezésének sorrendje

Ebben a „bőséges" helyzetben *(több kártya, mindegyiken több DNS-kiszolgáló)* a lekérdezések az alábbiak szerint alakulnak:

- A Windows kérést küld az elsődleges csatoló elsődleges DNS-kiszolgálójának
- Ha 1 másodperc múlva nem érkezik válasz, a Windows a kérést elküldi az összes hálózati kártya elsődleges kiszolgálóinak
- Ha 2 másodpercen belül sem kap választ, a kérést újra postázza, ezúttal már az összes csatoló minden DNS-szervere felé
- Ezután ha kell, további 2, majd 4 másodperc múlva újra elküldi a kéréseket, és az utolsó kérés elküldése után még 8 másodpercet vár a válaszra mielőtt a kérést sikertelennek nyilvánítja.



♣ ***A kiszolgálók lekérdezési sorrendje több csatoló és több DNS-szerver esetén***

Látható tehát, hogy az 1-2-2-4-8 másodperces késleltetési „lánc" ilyenkor is működik. Van azonban néhány apró szabály, ami a fenti táblázatot módosíthatja.

- Ha a nyolc másodperces várakozási idő végéig egy kártyán egyetlen DNS-kiszolgálótól sem érkezik válasz *(sem pozitív, sem negatív)*, Windows 2000 Professional *(XP)* esetén a Windows harminc másodpercig azon a kártyán nem kísérel meg további lekérdezéseket
- Ha egy DNS-kiszolgálótól érkezik válasz, de az negatív *("nem ismerem ezt a címet")*, a Windows a kártya további DNS-kiszolgálóit már nem kérdezi - hiszen ők is ugyanezt a választ adnák
- A kiszolgálók lekérdezésének sorrendje változhat, mert a Windows méri a válaszidőket és a legközelebbi esetben a leggyorsabb kiszolgálóhoz fordul majd először

Alhálózat-prioritás az ügyfélnél

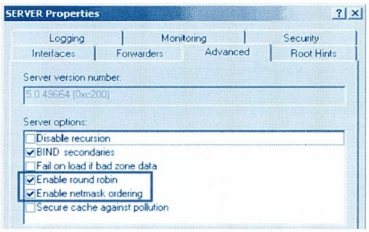
Ha a DNS-kiszolgálótól kapott válaszban egynél több IP-cím található *(ez gyakran előfordul)*, a Windows a címek közül kiválasztja azt, amely az ügyfél valamelyik hálózati csatolójával egy alhálózaton belül található. A művelet *(alhálózat-prioritás, angolul subnet prioritization)* csökkenti a hálózati terhelést, hiszen ha arra a lehetőség adott, az ügyfelek elsősorban a saját alhálózatukban *(azaz hozzájuk legközelebb)* található kiszolgálóhoz, IP-címhez fognak csatlakozni. Ezt a funkciót letilthatjuk, ha a registryben a

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\
Services\Dnscache\Parameters
```

kulcs alatt létrehozunk egy **PrioritizeRecordData** DWORD értéket, és azt 0-ra állítjuk. Ilyenkor a Windows azt az IP-címet fogja használni, amely a kiszolgáló válaszában a lista legelején állt.

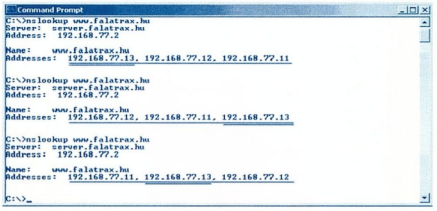
Round Robin

Térjünk vissza most egy kicsit a DNS-kiszolgáló beállításaihoz. A „Round Robin" egy angolszász gyerekjáték neve; hasonlít a mi „hátulsó pár előre fuss" játékunkhoz, csak ebben a játékban nem párok futkosnak, és a futás iránya is pont ellentétes: a gyerekek ott elölről hátulra futnak. Hogy mi köze van ennek a DNS-hez?



♣ A „körjáték" a kiszolgálón gyárilag engedélyezve van

A körjátéknak akkor lesz értelme, amikor a DNS-zónában egy névhez egynél több IP-cím tartozik. Bár a kiszolgáló válaszában ilyenkor mindegyik cím szerepelni fog, felmerül a kérdés, hogy milyen sorrendben - ugyanis az ügyfél nagy valószínűséggel a lista elején szereplő címet fogja majd használni.



♣ A Round Robin hatása a lekérdezésekre

Figyeljük meg, hogy az ismételt lekérdezések során hogyan változik a válaszban szereplő IP-címek sorrendje! Ezzel a módszerrel bizonyos szinten biztosítjuk a terhelés elosztását a kiszolgálók között. Ha a Round Robin opciót letiltjuk, az IP-címek sorrendjét az határozza meg hogy azok milyen sorrendben szerepelnek a zónafájlban, illetve az Active Directoryban.

Alhálózat-prioritás a kiszolgálón

Az ábrán bekereteztünk egy másik opciót is: a kiszolgálón már Netmask Ordering a neve, de valójában az előbb már megismert alhálózat-prioritásról van szó, mégpedig a kiszolgáló oldalán. Ha engedélyezve van *(és gyárilag igen)*, a kiszolgáló a válaszban visszaadott IP-címek sorrendjét módosítja, attól függően, hogy a kérdés küldő IP-címe mi volt. Tegyük fel, hogy a DNS-zónában a www.falatrax.hu IP-címei most az alábbiak:

```
www     10.0.0.11
www     172.16.0.11
www     192.168.77.11
```

A kérést küldő ügyfél IP-címe 192.168.77.101. Ha a kiszolgálón letiltjuk az alhálózat-prioritást, de a Round Robin engedélyezve van, a két egymást követő kérdésre adott válasz így fest:

```
Name: www.falatrax.hu
Addresses: 172.16.0.11, 10.0.0.11, 192.168.77.13

Name: www.falatrax.hu
Addresses: 10.0.0.11, 192.168.77.13, 172.16.0.11
```

Ha viszont hagyjuk az alapértelmezést, a kiszolgáló a válaszlista tetejére helyezi azt az IP-címet, ami az ügyfél IP-címével egy alhálózatban, illetve ahhoz „legközelebb" található. Ebben az esetben a két egymást követő válasz:

```
Name: www.falatrax.hu
Addresses: 192.168.77.13, 10.0.0.11, 172.16.0.11

Name: www.falatrax.hu
Addresses: 192.168.77.13, 172.16.0.11, 10.0.0.11
```

Látható, hogy a lista elején mindig az ügyfél alhálózatában szereplő cím szerepel. Észrevehetjük azt is, hogy a további címek sorrendje viszont változik, azaz a Round Robin, mint másodlagos rendezési elv, továbbra is érvényes.

Fülöp Miklós
mick@netacademia.net

Windows 2000 Active Directory telepítése

Mint az köztudott, a tartományvezérlői szerep *(az Active Directory)* nincs olyan módon bebetonozva a Windows 2000 Serverbe, mint ahogyan régen, azt NT4-es időkben *(PDC-BDC)* volt. Az AD is „csak egy" utólag telepíthető szolgáltatás, s ha úgy döntünk, akár el is távolíthatjuk a kiszolgálóról.

A mai vállalati hálózatokban az Active Directory olyan alapszolgáltatás, amely nélkül sok feladat nehézkesebben, vagy egyáltalán nem oldható meg. Mondhatják erre, hogy nem igaz, mert maximum nem-Microsoft technológiát használunk, de kérdem én: milyen nem-Microsoft megoldással lehetne gazdaságosan bevezetni a munkaállomásokon mondjuk a Kerberos-bejelentkezést? AD tehát kell. De hogyan tegyünk szert rá?

Minimális AD tervezés

Tekintettel cikkem témájára, az AD telepítésére, csak néhány tervezési szempontot veszünk górcső alá, s inkább a telepítés viszontagságaira összpontosítunk. Első és legfontosabb eldöntendő kérdés a tartomány neve, mert a DNS-integráció miatt ez sok minden mást, például leendő Active Directorynk DNS-fában elfoglalt helyét is eldönti. Nagyon fontos tervezési szempont, hogy utólag az AD tartomány neve nem módosítható másképpen, csak a teljes AD újratelepítésével. Egy újratelepítés viszont magával hozza *(rántja)* az Exchange 2000, a Certificate Services és még ki tudja, mi minden újratelepítését - tervezzünk tehát jövőbelátó módon!

A DNS-névtér meghatározásával megadjuk - később talán nagyra növő - AD-hierarchiánk csúcsát. Az AD-hierarchia építőelemei a tartomány *(egynél több tartományvezérlővel)*, a tartományokból felépíthető fa, és több fa összessége, az erdő. Ha csak egyetlen tartományt telepítünk, az előző fogalmakkal akkor is tisztában kell lennünk, mert

- Egyetlenegy tartományvezérlőnk egy nagyobb egység, a tartomány része.
- Egyetlen tartományunk egy nagyobb egység, a tartományi fa része, még akkor is, ha ennek a fának se ága se boga nincsen.
- Egyetlen fácskánk egy nagyobb egység, az erdő része. Ez még akkor is igaz, ha az erdőt pusztán a mi facsemeténk alkotja.

A fentiekből következik, és talán így már nem is igazán őrült meglepetés, hogy az Active Directory tartománytelepítő varázsló meg fogja kérdezni, hogy a parkerdőgazdaság területén hova kívánjuk elültetni a magocskát.

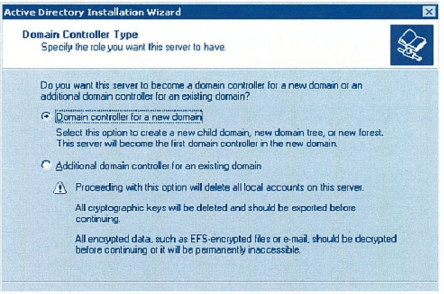
Ami a tartomány nevét illeti: az AD-nek nemcsak DNS-neve van, hanem - kompatibilitási okokból - régi típusú, NetBIOS neve is. Érdekes módon még a legmodernebb oprendszerek *(XP)* bejelentkezési ablakában is a NetBIOS-név virít, így ennek átgondolt kitöltése nem is olyan buta ötlet.

Általános szabály nincsen a jó tartománynév-képzésre, de jó ötlet DNS-névként az Interneten már bejegyzett név alatti tetszőleges altartományt elfoglalni *(pl.: bejegyezve: falatrax.hu, AD-tartománynév: ad.falatrax.hu)*, míg NetBIOS-névként a tartományra utaló nevet *(pl. FALATRAX)* választani. Ebből nem káosz következik, hanem olyan elnevezési konvenció, ami sohasem fog összeütközni az Internetes, publikus nevekkel, de nem kényszeríti a felhasználókat számukra értelmetlen nevek megtanulására.

Ennyi előkészület és tervezés után beleválhatunk a telepítésbe. Az Active Directory ugyanis - hasonlóan mondjuk az SQL Serverhez - egy utólag telepített „alkalmazás", amit kívánság szerint feltehetünk és levehetünk a kiszolgálókról. *(Rendben, legyen igazuk: elég különleges „alkalmazás" az, ami lecseréli a SAM-címtárat. De nem az AD az egyetlen, ami erre képes!)*

A DCPROMO

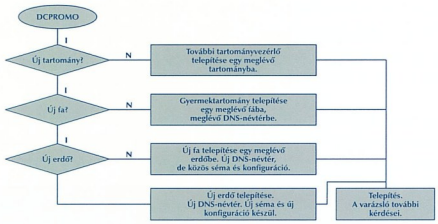
Az AD-telepítőt nem találjuk meg a Start menüben. Ennek nyilvánvaló oka, hogy nem szeretnénk, ha kósza felhasználók egere rátalálna. Bent van viszont a rendszergazdák számára megjelenő feladatlap elemei között, de mi nem innen indítjuk, mert tudjuk, hogy a program neve: DCPROMO.EXE. Indítsuk el! A semleges üdvözlőképernyő utáni első kérdés így „hangzik":



♣ *Új tartományt hozunk létre, vagy csatlakozunk egy meglévőhöz?*

Érdemes komolyan venni a sárga *(nyomtatásban szürke)* felkiáltójeles háromszög figyelmeztetéseit: ha nem új tartományt hozunk létre, hanem csatlakozunk egy meglévőhöz, az ezen a gépen, a SAM-ban tárolt felhasználók nem kerülnek be az Active Directoryba, hanem el fognak tűnni. Ezzel egyidejűleg a Protected Storage tartalma, vagyis a felhasználók RSA kulcspárjainak privát tagja is elillan. Ezt a kulcsok exportálásával *(mentésével)* előzhetjük meg. Ilyenkor szokott kiderülni, hogy a kulcsok non-exportable típusúak... Hát igen. A nyílt kulcsú infrastruktúra bevezetését jobb helyen tervezés előzi meg. Mi célból indítottuk a DCPROMO.EXE-t? Csak nem tartományvezérlőt szeretnénk telepíteni? Ha igen, vajon ez a gép egy meglévő tartományba kerül be sokadik vezérlőként, vagy egy új tartományt hozunk létre?

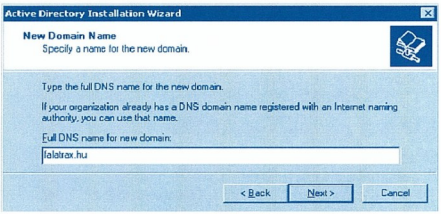
Hogy ne kelljen idemásolnom az összes hasonló kérdés képernyőképét, a DCPROMO kérdéseit stílszerűen egy fára fűztem fel, amit ezennel meg is mutatok:



♣ ***Az AD-telepítés lehetséges változatai***

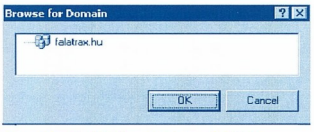
Tételezzük fel, hogy még nincs Active Directory a hálózatban. Ebben az esetben nyilvánvalóan új tartományt hozunk létre, de - a fentieknek megfelelően - egyben új fát és új erdőt is. Ennek értelmében a döntési fán végig az Igen ágon hajtunk végig. A telepítés folyamán az új erdő születésének részeként új séma és konfigurációs partíció fog kialakulni, amely az adott erdőben minden tagkiszolgálón közös lesz.

Miután eldöntöttük, hogy milyen telepítési változatot futtatunk *(új erdő)*, a telepítő megkérdezi a tartomány DNS-nevét. Aki huszadszorra futtataja a DCPROMO-t, talán észreveszi, hogy még azonos adatok megadásakor sem mindig ugyanúgy viselkedik. Új erdő telepítésénél például a DNS-nevet be kell gépelni. Nincs kiválasztást segítő lista *(a képernyőképet terjedelmi okokból meghamisítottam, összelapítottam)*:



♣ ***Az új tartományfa (és erdő) DNS-nevének megadása.***

Érdekes módon ha nem új tartományt telepítünk, hanem csatlakozunk egy meglévőhöz, a DNS-név kiválasztására segédablakot használhatunk, amely egyben a DNS-beállítások helyességének gyors ellenőrzésére is alkalmas, hisz ha üres a lista, valami nem stimmel a DNS-sel:

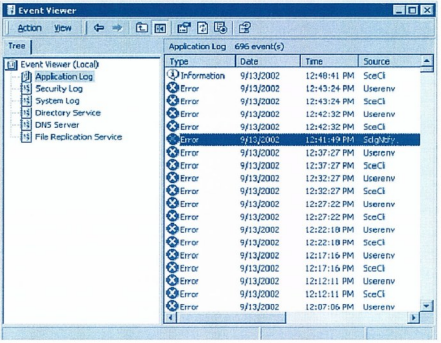


♣ ***DNS-név kiválasztását segítő lista***

Apropó, DNS!

Ha valami cigányútra viheti az Active Directory telepítését, az a hibás DNS-beállítás. Miközben valószínűleg minden bátor vállalkozó tisztában van az Active Directory DNS-függésével, valamint azzal hogy SRV-rekordokat jegyez be a zónafájlba a szolgáltatások elérhetőségének biztosítására, a DNS helyes beállítása gyakran elmarad.

Az átgondolatlan telepítések *(mint ahogy én is csináltam e cikk kedvéért)* kellős közepén kénytelenek vagyunk meghozni egy igen lényeges döntést: hol legyen a zónafájl? Ha ezt nem, vagy rosszul határozzuk meg, a DCPROMO után újrainduló gépen egy félszemű-féllábú Active Directory fogad majd minket. Szerencsére a DNS-hibák utólagos korrigálása teljes gyógyulást hoz - ennek elenére sokan újratelepítik az AD-t, mert nem tudják, hogy minden AD-hiba DNS-hibára vezethető vissza. Itt van például az általam a cikk kedvéért frissen telepített AD, mely olyan csonka lett, hogy a Group Policy editort nem lehetett elindítani *(mert nem találta a PDC-emulátort, vagyis saját magát)*, nem fogadott be további tartományvezérlőket és ráadásul az Application Logot telesírta a bánatával:



♣ ***Tipikus DNS-hiba, még ha a hibaüzenetek nem is erről tanúskodnak. Itt az áll, hogy a Group Policyt nem lehet kiértékelni***

A helyzet felismerésében óriási szerepe van a tapasztalatnak. Hajdanán biztosan újratelepítettem volna. Mai fejjel viszont pontosan tudtam, hogy a hiba oka DNS, csak meg kell találni, hogy mi a csuda. És meg is lett!

Ebben a számítógépben két hálózati kártya volt, melyek közül az egyiket nem dugtam be. Ilyenkor a legtöbb kártya Disconnected állapotba kerül, tehát nem sok vizet zavar - ám ez nem így viselkedett. Combo létére ugyanis akkor is él valamennyire, ha az UTP-csatlakozó üresen tátong. IP-címet is kap az APIPA-címtartományból *(169.254.x.y)*, melyet lelkesen és dinamikusan bejegyez a DNS-be - de érdekes módon nem hallgat rá.

A hiba oka röviden tehát annyi volt, hogy a gép két IP-címet jegyzett be a DNS-be, de ezek közül csak az egyikre válaszolt saját magának - viszont a DNS-kiszolgáló alapértelmezésben a két Host-rekordot felváltva adta vissza *(Round Robin)*: hol a jót, hol a rosszat!

A makrancos kártya letiltása, és némi DNS-pucolás után az Active Directory az összes baját elfelejtette, mankóját a sarokba hajította!

A DNS-infrastruktúra kialakítása

Az AD-nak szüksége van egy DNS-kiszolgálóra, melynek támogatnia kell az úgynevezett SRV-rekordokat. Hol találunk ilyet? A Windows 2000 telepítőlemezén. Hová telepítsük? Kézenfekvő módon a tartományvezérlői feladatra kiszemelt gépre, melynek jelenlegi TCP/IP-beállítása valami ilyesmi:

```
IP: 172.16.0.113
SM: 255.255.255.0
DG: 172.16.0.254
DNS: 172.16.3.8
```

A gép belső IP-címet, és egy belső, valahol a vállalati hálózaton lógó DNS-kiszolgálót használ. *(Ha nincs kitöltve a DNS-cím, az AD-telepítő felajánlja, hogy az adott gépre odavarázsol egy DNS-kiszolgálót.)* Ha most *(a Windowsban szokásos módon)* telepítjük a DNS Servert, és végignyomkodjuk a DCPROMO-varázslót, zátonyra futunk.

Ennek az az oka, hogy operációs rendszerünk minden esetben azt a DNS-kiszolgálót használja, amit a TCP/IP-beállítás meghatároz *(a fenti példában 172.16.3.8)*, függetlenül attól, hogy az adott gépen fut-e DNS Server, vagy sem. Az AD-telepítések hajnalán a hazai Internetszolgáltatók Linuxos DNS-kiszolgálói naponta több tucat DDNS-bejegyzést kaptak a magyar nagyvállalatok színe-javától a legkülönbözőbb fantázianevű AD-tartományok nevének bejegyzésére :-)

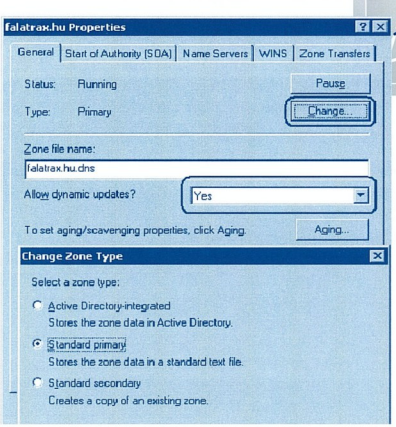
Tanulság: nem elég DNS-kiszolgálót telepíteni, a TCP/IP-paraméterek megfelelő átállításával használatba is kell venni azt!

Zónafájl létrehozása

Ha fent van a DNS-kiszolgáló, indítsuk el a megfelelő MMC-konzolt, és hozzunk létre egy úgynevezett Forward Lookup zónát, melynek neve egyezzen meg az általunk kiválasztott AD-névvel *(pl. falatrax.hu)*. Ezután állítsuk át a zónát dinamikusan frissíthetővé *(jobbklikk a zónán, tulajdonságok)*. Ha a zóna típusát a Change... nyomógomb segítségével Active Directory-integratedre állítjuk, a dinamikus frissítési lehetőségek között megjelenik a biztonságos *(secure)* frissítés is, melyről korábbi számainkban már többször volt szó.

Tipp

Ha - például politikai okokból - a dinamikus frissítés nem engedélyezhető, a DCPROMO után importáljuk be a zónafájlba a NETLOGON.DNS tartalmát. Ez a fájl ugyanis pontosan azokat a rekordokat tartalmazza (BIND-kompatibilis formátumban), melyeket az induló Active Directory bejegyzett volna.



♣ A zónát állítsuk át dinamikus frissítésűvé, hogy az elinduló Active Directory be tudja írni az SRV-rekordokat.

De most egyelőre haladjunk tovább a telepítés lépésein, mivel a rekordok regisztrációja a DCPROMO futása, majd a gép újraindítása után történik meg. Addig azonban van még egy-két kitöltendő mező...

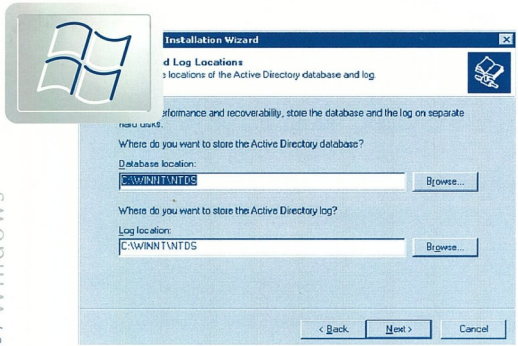
A NetBIOS-név

A tartomány DNS-nevének megadása után következik a NetBIOS-név kitöltése. Ne áltassuk magunkat: a NetBIOS-név még tisztán Windows 2000 környezetben is létfontosságú, sőt, gyakrabban találkozik vele felhasználó, mint a DNS-névvel! Tartományi bejelentkezés? Ott a NetBIOS-név! A hálózat erőforrásainak tallózása *(browse)*? NetBIOS-név alapján!

Ebből következik, hogy ennek értelmes megadása legalább olyan fontos, mint a DNS-név megtervezése. Ne írjunk DNS-névnek értelmetlen szavakat, trágár kifejezéseket stb.

Az Active Directory adatbázisai

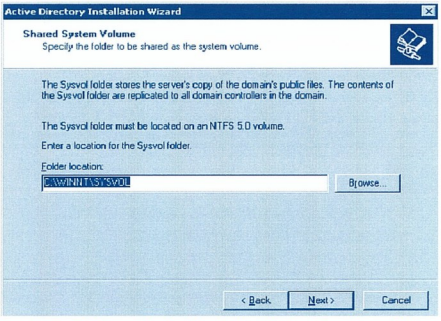
A hálózati nevek megadása után következik a címtáradatbázis elhelyezése. Az Active Directory adatai már nem a regisztrációs adatbázisban találhatók, hanem saját, tranzakciós naplóval védett Jet adatbázisban foglalnak helyet. A telepítő arra kíváncsi, hová tegye az adatbázisfájlt, valamint a tranzakciós naplót. Nagy teljesítményigény esetén a tranzakciónaplót érdemes külön lemezre helyezni. Mit nevezünk nagy teljesítményigénynek? A magyar vállalatoknál előforduló terhelést semmiképpen. Még ha napi száz embert vesz is fel, vagy rúg ki egy cég, akkor sem ér még közelébe sem az AD terhelhetőségének: a Compaq szakemberei több százmillió objektummal is megkínozták, mégsem fulladt meg! Meghagyhatjuk tehát az alapértelmezett útvonalakat - ha szükség van rá, szerencsére a későbbiekben bármikor átköltöztethetjük a címtárat más kötetre, más útvonalra.



♣ *Az Active Directory adatbázisának és tranzakciónaplójának helye*

A SYSVOL-kötet

Mit tároljunk a címtárban? Mindent bele? A dolgozók fényképei, személyi száma belekerüljenek? Vizsgáljuk meg, a Microsoft hol húzta meg a határt: a redmondi fiúk fájlokat nem pakolnak az AD-ba. Szép dolog az egységes címtár, de vannak adatok, melyeknek nincs helyük az AD hierarchikus LDAP tárolójában. Érdekes módon maga az AD is rendelkezik olyan adatokkal, melyeket nem önmagában hordoz. A csoportos házirend például ilyen. A beállítások kisebbik része az AD-ban bújik meg, a többség *(logon/logoff scriptek, administrative template-ek stb.)* azonban fájlokban csücsül. Hol? A SYSVOL-könyvtárban:

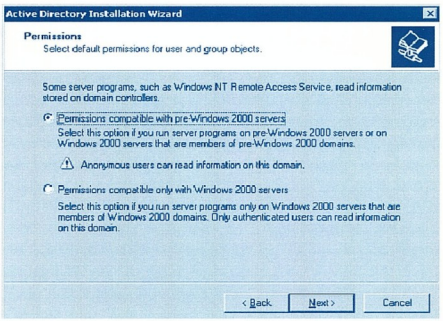


♣ *A SYSVOL-könyvtár helyének beállítása. Ennek replikációját nem az AD, hanem a File Replication Service végzi majd!*

A SYSVOL tartalmának replikációjáért nyilvánvalóan nem az AD felel *(hisz az csak saját adatbázisát másolgatja szorgosan a többi tartományvezérlőre)*. A Windows 2000 szolgáltatásai között találjuk a File Replication Service *(FRS)* komponenst, mely alapértelmezésben csak és kizárólag a SYSVOL-t másolgatja távoli tartományvezérlőkre, de szerepet kap az elosztott fájlrendszer *(Distributed File System, DFS)* kópiáinak kezelésében is.

Jogosultságok a címtáron. Ki olvashatja?

A DCPROMO következő kérdése a címtár alapértelmezett jogosultságainak kezdeti beállítását feszegeti. Vajon lehetővé tegyük-e mindenki számára, hogy olvassa a címtár tartalmát? Az ábra tanúsága szerint például az NT4 tagkiszolgálókon futó RRAS-oknak lehet szükségük az Anonymous címtárolvasásra. Több példám nekem sincs, de ha nem akarjuk kockáztatni NT4-en futó alkalmazásaink életbenmaradását, válasszuk a fenti opciót.



♣ *Kilyukasztjuk a címtárat? Tegyünk a gyökerére az Everyone csoport tagjai számára olvasási jogot?*

Bármelyiket választjuk is, legközelebb ez az opció érdekes módon az Exchange 2000 Server telepítésénél köszön vissza. Az Exchange azt is elárulja, hogyan tehetjük biztonságosabbá a címtárat *(egyébként az Everyone - Read jog eltávolításával)*.

Van SAM? Nincs SAM?

Az Active Directory telepítésével megszüntetjük a hitelesítési feladatok korábbi ellátójának, a SAM-nak szerepét. De SAM nem hal meg, csak háttérbe vonul! Mindaddig nem jut szóhoz, amíg az Active Directory fut. Ám ha ez utóbbi nem indul el, a hitelesítési feladatokat ismét SAM látja el. Ez akkor fordulhat elő, ha az Active Directory valamilyen fatális baleset folytán, vagy szándékos beavatkozás következtében *(Safe Mode, Recovery Console)* nem áll rendelkezésre. A bejelentkezés ilyen esetekben is kötelező - jön hát SAM.

Igen ám, de milyen felhasználók állnak rendelkezésre SAM szerint? Leginkább csak az Administrator. És vajon mi a csuda a jelszava? Most tessenek kapaszkodni: az, amit a DCPROMO következő képernyőjén adunk meg!

Mielőtt az AD átvenné a hitelesítési feladatokat SAM-tól, még beállítjuk a terepről éppen levonuló Administrator jelszavát. Erre a legritkább esetben lesz majd szükség, de amikor igen, hirtelen nem fog eszünkbe jutni ez a jelszó! Írjuk fel olvan helyre, ahol biztonságosan megőrizhető a következő katasztrófa bekövetkeztéig!

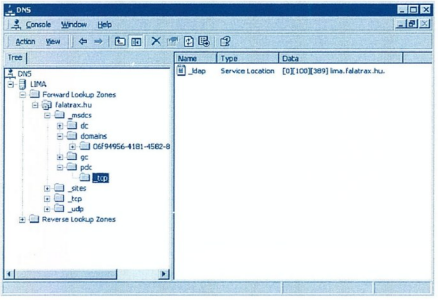
♣ *A SAM-ban tárolt Administrator fiók jelszavának beállítása - a nehéz napokra*

Több kérdés nincs, a telepítés végre-valahára beindul. Néhány percnyi aktivitás után a varázsló felszólít a gép újraindítására, s ha szerencsénk van, egy AD-tartományvezérlő kel életre. A sunyi DNS-hibák miatt azonban mindenképpen érdemes végignézni az új rendszert, hogy minden rendben van-e.

A telepítés ellenőrzése

Újraindítás után jelentkezzünk be rendszergazdaként, és elsőként vizsgáljuk meg az Active Directory adatbázist a merevlemezen: ottvan-e? Ehhez látogassunk el a C:\WINNT\NTDS könyvtárba, és keressük meg az NTDS.DIT fájlt. Ez az adatbázisfájl. E mellett találhatók a tranzakciós naplófájlok *(*.log)*, mindegyikük kereken 10 MB. Ha át szeretnénk helyezni máshová, a parancssorból indítható NTDSUTIL programocska segít nekünk.

Második ellenőrzési pontunk a DNS-zóna legyen. Benne vannak az SRV rekordok? Egy kerekre kitömött zónafájl az alábbi ábrához hasonlóan néz ki:



♣ Jólápolt Active Directory-zónafájl

A falatrax.hu alatt az alábbi altartományokat találjuk:

- _msdcs: Microsoft Domain Controllers. Ez tovább bontható dc, domains, gc és pdc tartományokra. A gc jelentése: Global Catalog, a pdc-bejegyzés pedig PDC-emulátorra utal.
- _sites: itt ugyanazokat a tartományvezérlőket telephelyek szerinti bontásban találjuk, ez alapján találják meg a munkaállomások a hozzájuk leközelebb eső kiszolgálókat
- _tcp és _udp: az egyes szolgáltatások felbontása TCP-csatorna-elérési szempontból

Amennyiben ezek a bejegyzések hiányoznak, biztosra vehető, hogy az AD még alapfunkcióit sem fogja tudni normálisan ellátni. Ha nincs meg minden, vagy üres a zóna *(mert például elfelejtettük engedélyezni a dinamikus frissítést)*, a következő trükk segíthet:

```
NET STOP NETLOGON

NET START NETLOGON
```

A NetLogon szolgáltatás felelős a tartományi SRV-rekordok regisztrációjáért, amit induláskor végez el. A Host *(A)* rekordok erőszakos bevésésének módja pedig a következő:

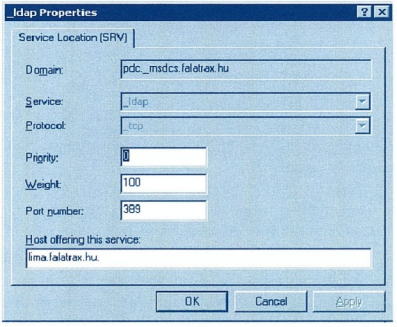
```
IPCONFIG /REGISTERDNS
```

SRV-rekordok

Mik is tulajdonképpen az ezerszer is említett SRV-rekordok? Nos, leginkább az NS és MX-rekordokhoz hasonlítanak, mivel:

- egy konkrét szolgáltatás *(szolgáltatási pont)* elérésében segítenek,
- indirekt címzést használnak *(nem IP-címre mutatnak, hanem „A" vagy „CNAME" rekordra)*
- egynél több azonos rekord esetén prioritást lehet felállítani a szolgáltató kiszolgálók elérésére

Az SRV nem más, mint az IEFT *(Internet Engineering Task Force)* válasza a gyártók egyre sokasodó egyedi szolgáltatásrekord-bejegyzési kérelmére. Ahelyett, hogy lenne külön Kerberos-rekord, GC-rekord és hasonlók, az IETF egy általános bejegyzési módszert alkotott, mellyel gyártó- és RFC-függetlenül végezhetjük el a sok-sok sajátos szolgáltatás bejegyzését. Egy SRV-rekord felépítése a következő:



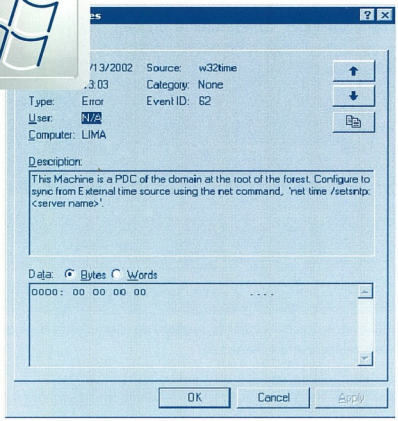
♣ A PDC-emulátor kiszolgálószerep az SRV rekordoknak köszönhetően zökkenőmentesen felvehető a DNS-be

- Tartalmazza az adott szolgáltatás végzéséért felelős gép FQDN-jét,
- az adott szolgáltatás megnevezését *(pl.: _ldap, _kerberos, _smtp)*
- a kapcsolattartás módját *(_tcp vagy _udp)*
- az adott rekord prioritását,
- az azonos prioritású, azonos rekordok közötti terhelés-eloszlás arányszámát *(weight)*
- és a szolgáltatás eléréséhez használandó port számát *(LDAP-nál 389, SMTP-nél 25 stb.)*

Tulajdonképpen az SRV-rekordok megjelenésével a korábbi szolgáltatásrekordok feleslegessé váltak, hisz ezzel az általános megközelítéssel mind az NS, mind az MX rekordok leírhatók lennének - de hát a kompatibilitás miatt örökre fent fognak maradni.

További ellenőrzési pontok

Gondos gazda nem hagyhatja ki az eseménynaplót, amikor körbejárja a rendszert. Nicsak! Hiba van benne! Nem szép dolog egy frissen telepített rendszertől! Nézzük csak meg közelebbről!



♣ Az SNTP szolgáltatás hibát jelez: nincs beállítva a rendszeróra-szinkronizáció forrása!

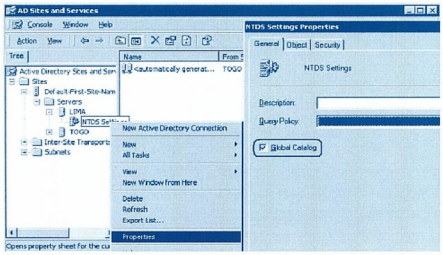
A hibaüzenet jelentése: ez a gép az erdő csúcsának PDC-emulátora. *(Azaz az összes további számítógép, legyen az DC vagy egyszerűen munkaállomás, innen fogja kapni a pontos időt.)* Állítsuk be egy külső időforráshoz, mert különben irgumburgum. Szerencsére a megfelelő parancsot is megkapjuk, így legalább ezen nem kell gondolkodni. De mi legyen a külső időforrás? Ha van Internetkapcsolatunk, legyen például ez, a Microsoft által üzemeltetett SNTP-kiszolgáló:

```
NET TIME /SETSNTP:time.windows.com
```

Ha nincs kapcsolat, a hibaüzenetet viszont nem szeretnénk látni a jövőben, állítsuk a „külső" forrást saját magára!

```
NET TIME /SETSNTP:127.0.0.1
```

Furcsán fest, de működik. Illetve nem működik, de legalább nem kiabál.
A következő ellenőrzési pont a bejelentkezéshez szükséges kiszolgálószerepek meglétének ellenőrzése. A legeslegfontosabb a Globális Katalógus megléte. Ennek ellenőrzéséhez indítsuk el az Active Directory Sites and Services eszközt, majd ássunk le az alábbi ábrának megfelelő módon:

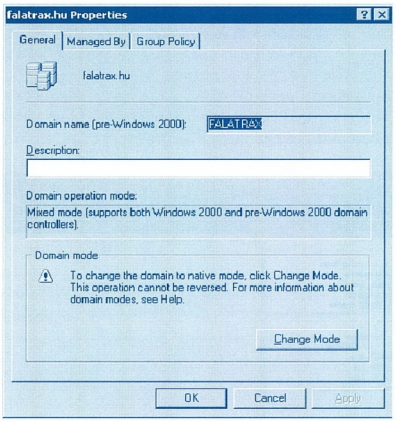


♣ Az AD Sites and Services eszközzel lehet bállítani, melyik tartományvezérlők tetszelegjenek Global Catalog szerepben?

Ha ez megvan, még mindig nem biztos, hogy működik. A Windows 2000 CD-n található Support\Tools eszközkészlet része a DCDIAG parancssori eszköz, mely megmondja, hogy üzemel-e a GC *(és a többi szerep)*, vagy sem. Ha nem, akkor sem kell csüggedni, hisz már mindent tudunk: a hiba oka biztosan a DNS!

Mixed vagy Native?

Egy vadonatúj AD-erdő telepítése után az ember azt gondolná, hogy a rendszer natív üzemmódban fut. És nem.

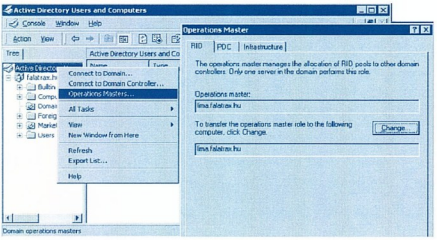


♣ A frissen telepített tartomány mixed üzemmódban fut

Ha soha nem fogunk NT4 BDC-t telepíteni ebbe a tartományba, bátran álljunk át mixed módról natívra, nem veszítünk vele semmit - igaz, nem is nyerünk szinte semmit. Indítsuk el az Active Directory Users and Computers eszközt, kattintsunk a jobb egérgombbal a tartomány nevén *(esetünkben falatrax.hu)*, válasszuk a tulajdonságok menüpontot, s a megjelenő ablakban ne féljünk a Change Mode nyomógombtól: az én praxisomban még sohasem okozott gondot a váltás!
És micsoda érzés, hogy natív üzemmódban a csoportok típusát menet közben át lehet állítani *(globális -> lokális -> univerzális)*, hogy a csoportokat körkörösen egymásba lehet ágyazni, hogy policyval szabályozhatjuk a RAS-hozzáférést, hogy több mint 40.000 objektumunk lehet...

FSMO-szerepek

Egyetlenegy tartományvezérlő esetén az összes speciális kiszolgálószerep - nyilván - ezen az egyetlen DC-n található. Körutunkat zárjuk azzal, hogy leellenőrizzük, amit amúgy is biztosan tudunk: ez a DC futtatja az összes FSMO-szerepet. *(A szerepekről később részletes cikket írok.)*
A gyakorlat annyiban mégsem haszontalan, hogy megtanuljuk, hol rejtőznek, és kik is tulajdonképpen a fizmók *(így ejtik ki az FSMO-rövidítést)*. Öt ilyen szerep van, ezek közül hármat az ADUC *(Active Directory Users and Computers)* mutat meg:



♣ *FSMO-szerepek megtekintése, átállítása ADUC-cal*

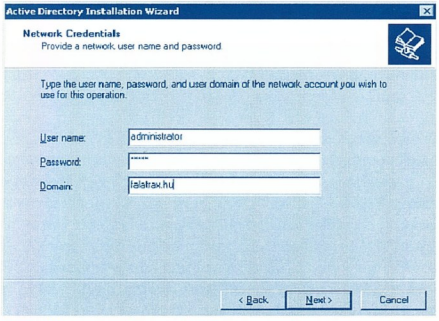
A fennmaradó kettőt azért nem itt találjuk, mert nem tartomány, hanem erdőszintűek *(a Domain Naming Master és a Schema Master)*.
Ha egynél több tartományvezérlőnk lenne, most átpasszolhatnánk e szerepeket egyik gépről a másikra.

Az AD-telepítés ökölszabálya

Bárki bármit mond, az Active Directory illékonynak tekinthető, ha csak egyetlen tartományvezérlőnk van. Tartományonként minimum kettőt illik üzemeltetni ahhoz, hogy mindig legyen egy fotóképes példányunk. Különösen igaz ez az aranyszabály az erdő gyökerére. Ha ugyanis ez eltűnik, az összes alatta lévő fa, cserje, bozót kivágható, felégethető - mert egy másik erdőt összerakni nem lehet belőlük.
Tehát következzen a második tartományvezérlő telepítése.

DCPROMO mégegyszer

A második gépet úgy telepítjük példánkban, hogy az a meglévő, falartax.hu tartománynak második DC-jévé váljon. A DCPROMO képernyői között felbukkan egy újabb, amely hitelesítési adatokat kér tőlünk:

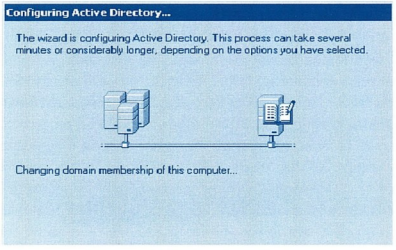


♣ *További tartományvezérlők és gyermektartományok csatlakoztatásakor meg kell adnunk egy erős ember nevét és jelszavát*

Itt természetesen a céltartományban érvényes név-jelszó párost kell megadni, mégpedig olyan felhasználóét, aki el tudja látni a következő feladatokat:

- létre tud hozni egy számítógépfiókot
- át tudja minősíteni DC-fiókká
- át tudja mozgatni a Domain Controllers tárolóba

Ez az ember nem más, mint az Administrator. Lehet ugyan csűrni-csavarni a jogosultságokat, de szerintem nem érdemes: DC telepítésre nem minden nap kerül sor.
A DCPROMO után már csak a kezdeti címtárreplikációt kell kivárni, és lesz két makkegészséges tartományvezérlőnk.



♣ ***A második DC kezdeti replikációja során átveszi az elsőtől a teljes címtárat: a sémát, a konfigurációs partíciót, a felhasználói adatokat***

Ötperces szinkronizációs idővel fognak replikálni mindaddig, amíg el nem magyarázzuk nekik, hogy milyen hálózati infrastruktúra húzódik alattuk. De ez már egy másik történet.

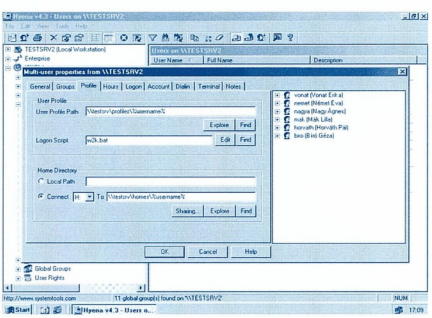
Fóti Marcell
marcellf@netacademia.net

Az AD szelidítés diszkrét bájai

Biztos nem vagyok egyedüli birtokosa annak a furcsa élménynek, amikor anno a Windows 2000 frissítés után több felhasználót az Active Directory-ban kijelölve hiába gúvadt ki a szemem a Properties menüpontot keresve. Nincs. Hát akkor? Muszáj lesz egyenként a 625 felhasználónál a „Password never expires" kapcsolót bebillenteni? És ha változik a Home mappák helye? Brrrr. A konzervatív, NT4-ről érkező rutinos rendszergazda megpróbálja becsapni önmagát, és nyomban indítja a váltás ellenére megmaradt User Managert, de - érthető módon - a vörös ködön átívelő piros keresztes hibaüzeneteken kívül semmi másra nem jut vele. Akkor jöjjön minden titkok tudója: az Internet! Technet, KB, Google susogja a javasolt megoldást: szkript, szkript, szkript, csvde, ldifde, ADSI. Jó, akkor adjunk egy esélyt a „Lifelong Learning" szellemének! Mégsem jó. Elsőre - és különösen kezdőként - nehéz szkriptekkel „uralni az AD-t", bonyolult, aprólékos, nincs rá idő, és nem tűnik túl barátságos lehetőségnek. Így aztán első körben a megoldást valószínűleg nem a programozó rendszergazda vonalon találjuk meg, hanem inkább a grafikus, külső alkalmazásoknál. Ezek közül válogattam ki rövid bemutatásra hármat.

Hyena

Bizarr neve ellenére a Hyena régóta kellemes megoldást nyújt egy NT4/Windows 2000 tartomány rendszergazdai feladatainak ellátásában illetve pl. a konkrét példánál maradva a több felhasználói fiók tulajdonságainak egyszerre történő megváltoztatásával kapcsolatban is. Kezelése egyszerű és *(megdöbbentő módon :))* megszólalásig hasonlít az MMC filozófiához: a bal oldali keretben a három kezelési szintet *(Local Workstation, Enterprise, Domain)* jobb oldalon az aktuális bal oldali szinthez tartozó részleteket láthatjuk. Így pl. ha kibontjuk a bal oldali tartományt, akkor a Users menüponton kettőt kattintva a jobb oldalon megjelenik az összes felhasználói fiók *(szervezeti egységektől függetlenül)* és ha a kívánatosakat kijelöljük, lesz User Properties menüpont!



♣ *Ami biztosan hiányzik a Windows 2000-ből...*

A program ezen kívül még számtalan lehetőséget ad a kezünkbe. Bűvészkedhetünk a felhasználói jogokkal, lekérdezhetünk egész különleges információkat *(tényleg 680 napja változtatta meg Józer Jolán utoljára a jelszavát???)*, kezelhetünk, exportálhatunk csoportokat, számítógépeket, felhasználókat és beállításokat, lehetőségünk van a Terminal Services-zel kapcsolatos tulajdonságok beállítására, de készíthetünk ezekre a feladatokra makrókat is, valamint pl. integrálhatjuk a felhasználó létrehozási folyamatba az Exchange 5.5/2000 ide kapcsolódó funkcióit is. Némiképp egyszerűbb és gyorsabb mint az *(első esetén)* az ADC-vel küzdeni! A program letölthető a **[1]** címről.

Dameware

A Dameware NT Utilites *(letölthető a **[2]** címről)* nem elégszik meg a Hyena komforszintjével, hanem annál többet nyújt. Ez rögtön nyilvánvalóvá válik, ha megnézzük például a kiszolgálókkal kapcsolatos műveletek listáját! Elképesztő mennyiségű parancsot, műveletet zsúfoltak össze a készítők. Van a programnak egy nagyon kellemes funkciója is, az „Add New User Multi", amellyel némi előzetes hangolás után másodpercek alatt akár többezer felhasználói fiókot hozhatunk létre. A csomag tartalmaz még egy Remote Control programot *(távvezérléssel lehet a klienst is felrakni!)* és egy „Service Install and Remove" segédprogramot is.

UserManagemeNT Lite

Ezt a programot kicsit nehézkesebb kezelni, viszont van egy hatalmas előnye a másik kettőhöz képest: teljesen ingyenes. A telepítés után egy 9 lépésből álló alapos környezet-konfigurálás következik, ha ezt végigcsináljuk, akkor az előzőekben említett programoknál kisebb tudású és némiképp eltérő felületen tudunk dolgozni. Megtalálható és regisztráció után letölthető a **[3]** címről.

Inkább mégis uraljuk?

Meglátásom szerint ezek az eszközök - bármennyire hasznosak is - elsősorban csak „első körös" megoldások. Ennek okai között szerepel elsősorban az a tény, hogy ugyan teljes funkcionalitásúak, de időkorláttal vannak ellátva, a végleges változatuk ára viszont eléggé borsos. Ehhez jön még az a körülmény, hogy az első benyomások ellenére azért az adminisztrációs szkript írása mégiscsak egy az elsajátítható „tudományágak" közül és mindenképpen előremutató gondolkodásra utal, ha otthon tudjuk magunkat érezni ezen a területen is.

Úgy tűnik - persze ez is csak fikció részemről -, hogy Redmond felé is érkezhetett némi igény ezeken a területeken, mert a .NET Server RC1-ben már újra van lehetőség több felhasználói fiók jellemzőinek egyszerre történő módosítására. Így aztán megint volt egy furcsa élményem :).

Gál Tamás (MCP)
gtamas@tjszki.hu

A cikkben szereplő URL-ek:
[1] http://www.systemtools.com/hyena
[2] http://www.dameware.com
[3] http://www.advtoolware.com/t4e/uml/uml_default.htm

DFS és FRS - Elosztott fájlrendszer és replikáció

A két hárombetűs rövidítés két, kevésbé ismert Windows 2000 rendszerszolgáltatást takar. A Distributed File System, az elosztott fájlrendszer a hálózati megosztott mappák közös névtérbe szervezésében segít nekünk, a File Replication Service *(fájlreplikációs szolgáltatás)* pedig a mappák tartalmának szinkronizálását végezheti el helyettünk.

Elosztott fájlrendszer? Közös névtér?

Miért van szükségünk az elosztott fájlrendszerre? A legtöbb számítógépes hálózatban a felhasználók adataikat központi kiszolgálók megosztott mappáiban tárolják. Minden ilyen hálózatban előbb vagy utóbb bekövetkezik az a kényes szituáció, hogy a megosztott mappákat másik, újabb, nagyobb teljesítményű kiszolgálóra kell mozgatnunk. Előfordul az is, hogy a régi szerver más célokra még a hálózatban marad, azaz a megosztott mappa nevében található szervernév megváltozik. Igen ám, de mi legyen a felhasználók gépein hemzsegő hálózati meghajtókkal, parancsikonokkal, beállításokkal, sok-sok hivatkozással, amelyek mind-mind a régi útvonalat tartalmazzák? Ezek egy része például a bejelentkezési parancsfájl segítségével módosítható, de biztos, hogy marad néhány olyan hivatkozás, ami később még komoly bosszúságot okoz. A közös névtér egyik előnye, hogy a megosztott mappák „valódi" útvonalát elrejti a felhasználók elől, így egy-egy ilyen változtatás nem okoz pluszmunkát az ügyfélgépek oldalán.

A DFS további előnyei

A Distributed File System szolgáltatás minden Windows 2000 Serverben megtalálható, bár alapértelmezésben nem fut. Lássuk, milyen előnyökkel jár még a DFS alkalmazása egy Windows hálózati környezetben:

- A közös névtér központilag, a DFS kiszolgáló*(ko)*n indítható konzol segítségével kezelhető
- A névtér megosztott mappákra mutató elemei, az úgynevezett DFS linkek több megosztott mappára is mutathatnak. Ilyenkor az ügyfélgép automatikusan kiválasztja, hogy melyik megosztást használja majd
- A DSF link, azaz megosztott mappa több példánya *(a „replikák")* közül az ügyfélgép véletlenszerűen választ, így a megosztott mappák között automatikus terheléselosztás alakul ki
- A Windows 2000 kliensek a használt replika kiválasztásánál előnyben részesítik azokat, amelyek velük közös telephelyen *(Active Directory site-ban)* találhatók
- Az „ügyfélalkalmazás" Windows 98-tól, illetve Windows NT 4.0 SP3-tól kezdődően minden Windows része, és minden további beállítás nélkül használható
- A replikáknak köszönhetően a DFS-névtérben az adatok többszörösen tárolódnak, így a rendszer hibatűrő marad. Cikkünk másik főszereplője, a Windows 2000 File Replication Service szolgáltatása ráadásul a mappák automatikus szinkronizációját is lehetővé teszi *(a DFS egy bizonyos üzemmódjában, a tartományi DFS módban)*
- Ugyanebben az üzemmódban a DFS névtér adatai az Active Directory-ban tárolódnak, így maga a névtér is hibatűrő, többszörözhető

A DFS szolgáltatás üzemmódjai

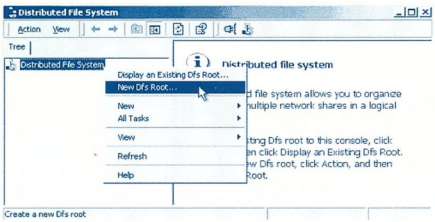
Az elosztott fájlrendszer-szolgáltatást két különböző üzemmódban indíthatjuk el:

- Stand-Alone DFS: azaz önálló DFS kiszolgáló. Ebben az esetben a DFS szolgáltatás a névtér információit a DFS kiszolgáló regisztrációs adatbázisában tárolja. Maga a névtér gyökere *(DFS root)* is a DFS kiszolgálón keresztül érhető el, ha ez a kiszolgáló nem elérhető, a DFS „halott"
- Domain DFS: magyarul tartományi DFS kiszolgáló. Ilyenkor a DFS szolgáltatás adatai értelemszerűen az Active Directory-ban találhatók. Tartományi DFS esetén is kell legalább egy kiszolgáló, ami a DFS gyökér egy példányát kiszolgálja, de ebben az üzemmódban maga a DFS gyökér is többszörözhető *(azaz gyökérreplikák is rendelkezésre állnak)* - persze minden replikának különálló kiszolgálóra kell kerülnie.

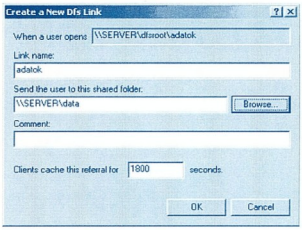
Üzemmódtól függetlenül szabály, hogy egy kiszolgálón csak egy DFS gyökér lehet, viszont a tartományon belül több DFS gyökeret is létrehozhatunk *(nyilván mindegyiknek másik kiszolgálóra kerül a replikája)*. A kétféle DFS szolgáltatás egyébként nagyon jól megfér egymás mellett, sőt, ezeket kombinálhatjuk is, de erről majd később.

Stand-Alone DFS telepítése

A DFS olyan könnyen eltávolítható, amilyen könnyen telepíteni *(tulajdonképpen csak elindítani)* is tudjuk, ezért bátran kísérletezgethetünk akár otthon is. Az alapvető funkciók bemutatása céljából először az egyszerűbb, önálló DFS szolgáltatás telepítését mutatjuk be. Ehhez szükségünk lesz egy Windows 2000 Serverre, amit kinevezünk DFS kiszolgálónak. *(Önálló DFS révén az Active Directory-ra, de még Windows tartományra sincs szükség)*. Indítsuk el a DFS felügyeleti konzolt az Administrative Tools menüből!



♣ *A DFS felügyeleti konzol*



♣ *DFS link létrehozása*

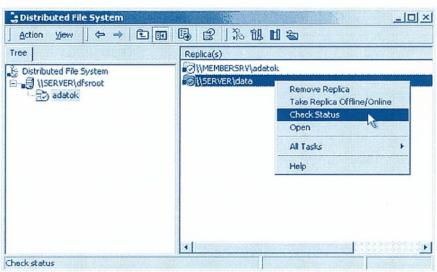
Tegyük fel, hogy két kiszolgálónkon egy-egy megosztott mappákban tároljuk adatainkat. A megosztások már léteznek, valódi elérési útjuk legyen például:

```
\\SERVER\data\
Øs
\\MEMBERSRV\adatok\
```

Jegyezzük meg, hogy önálló módban a mappák tartalmának szinkronizálásáról magunknak kell gondoskodnunk!

A DFS névtér létrehozása egy DFS gyökér „telepítésével" kezdődik. A DFS gyökér egy, a DFS kiszolgálóként üzemelő Windows 2000 Server egy speciálisan erre a célra megosztott mappáját is jelenti. Ha a DFS konzolban a New DFS Root... parancsot választjuk, elindul a DFS gyökér létrehozására készített varázsló. A varázsló első kérdése a létrehozni kívánt DFS típusa, itt válasszuk ki a „Create a standalone DFS root" opciót. Ezután meg kell adnunk a DFS gyökér kiszolgálóját (célszerűen azt, amelyen a konzol is fut), majd ki kell választanunk a DFS gyökér megosztott mappáját. A varázsló felajánlja nekünk a kiszolgálón pillanatnyilag élő megosztásokat, vagy segít nekünk újat létrehozni, ha úgy gondoljuk. A DFS gyökér „adatait" tartalmazó megosztott mappa illetve könyvtár egyébként helyet igazán nem foglal, csak a virtuális névtérnek megfelelő mappastruktúra kerül majd bele *(hiszen a valódi adatok a kiszolgálókon vannak)*. Egy hasznos tanácsot érdemes csak megfogadni: a DFS gyökérmappája NTFS partícióra kerüljön.
Az új könyvtár és megosztás létrehozásához tehát válasszuk a „Create new share" opciót, adjuk meg a könyvtár nevét *(pl. „C:\dfsroot\")*, és a megosztás nevét.
A megosztás nevének kiválasztásánál már legyünk körültekintőek, ugyanis a DFS névtér hivatkozásaiban a gyökérmegosztás neve már szerepel *(pl. „\\SERVER\dfsroot\")*. Ha a könyvtár még nem létezik, a varázsló létrehozza nekünk, majd még a megosztáshoz is fűzhetünk kommentárt. Ezzel a DSF gyökér létrehozása befejeződött. A fenti hivatkozást bármely Windows kliensbe bepötyögve máris hozzáférnénk az egyelőre még teljesen üres DFS fához.
A következő feladat a névtér benépesítése, amikor leveleket aggatunk a DFS fára. Most hozzuk létre a megosztott mappákra mutató hivatkozásokat *(esetünkben persze csak egyet)*, és beállítjuk azok replikáit is. Kattintsunk jobb gombbal az imént létrehozott DFS gyökérre, és a felugró menüből válasszuk a New DFS link... parancsot!

Adjunk nevet a linknek *(ezt látják majd a felhasználók, ahogy gépelünk, mi is látjuk a teljes elérési utat az ablak tetején)*, azután pedig adjuk meg az első megosztott mappát. Miután az OK gombra kattintunk, a bal oldali fában megjelenik a link, a jobb oldali listában pedig a linkhez tartozó replikák *(esetünkben egyelőre csak egy)*. Adjuk hozzá a másik kiszolgáló megosztott mappáját is! Ehhez kattintsunk jobb gombbal a fában a DFS hivatkozásra és a New Replica... parancsra megjelenő ablakba írjuk be a másik megosztott mappa elérési útját.



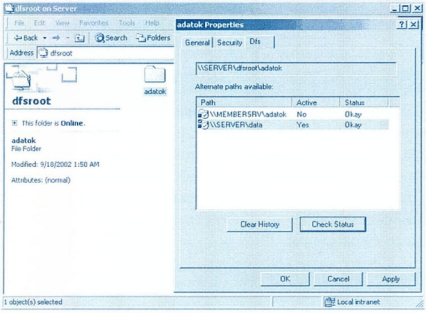
♣ *A kész DFS névtér. A Check Status paranccsal a megosztott mappák elérhetősége ellenőrizhető*

Miután az így készült hálózati elérési út *(UNC path)* bárhol használható, ahol egy „valódi" elérési utat is megadhatunk, nincs akadálya annak sem, hogy egy DFS link egy másik DFS névtér egy elemére mutasson. Így szép kis hierarchiát építhetünk *(egyébként önálló és tartományi névterekből vegyesen is)*. A Windows ügyfél a DFS hivatkozásokat képes szépen követni. Éppen ezért vigyázzunk is: óvakodjunk a DFS névterek közötti kereszthivatkozásokról, ugyanis a Windows nem ismeri fel a rekurziót, és adott esetben a végtelenségig *(de legalábbis a rendelkezésére álló erőforrások erejéig)* keresné a „valódi" elérési utat - mindhiába.

A DFS ügyfelek

A cikk bevezetőjében említettük, hogy Windows 98-tól kezdve beépítetett DFS ügyfél található minden Windows operációs rendszerben. A valósághoz tartozik az is, hogy a DFS ügyfélalkalmazás letölthető Windows 95-höz is. A Windows 9x azonban csak olyan DFS linkekhez képes csatlakozni, amelyek Windows kiszolgálón megosztott mappákra hivatkoznak. Az NT, a Windows 2000 és utódaik minden további nélkül megbirkóznak azzal is, ha a DFS hivatkozás éppen egy Netware vagy NFS megosztásra mutatnak.
A DFS link létrehozásakor megjelenő ablakban *(és a link tulajdonságlapján is)* láthatunk egy értéket, ami azt határozza meg,

hogy az ügyfélgépek mennyi ideig tárolhatják a DFS hivatkozásokat a helyi gyorsítótárukban. Minden viszszafejtett DFS hivatkozás belekerül ebbe a gyorsítótárba, és ha az előre beállított idő alatt az adott DFS linket újra vissza kellene fejteni, az ügyfél nem fordul a DFS kiszolgálókhoz. Ez az érték alapértelmezésben 1800 másodperc, azaz fél óra, ami természetesen átállítható.
A Windows 2000 és XP ügyfél, amellett hogy a megfelelő replika kiválasztásánál figyelembe veszi a telephely-információkat is, grafikus felülettel is rendelkezik: minden DFS mappa tulajdonságlapjára egy új, „Dfs" feliratú fül kerül:



♣ *A DFS mappa plusz tulajdonságlapja Windows 2000 (és XP) alatt*

Ezen az oldalon láthatjuk, hogy a DFS hivatkozás milyen replikákat tartalmaz, hogy ezek közül a gépünk melyiket választotta (az *„Active" oszlopban itt „Yes" található)* - de elég kettőt kattintanunk egy másik sorra és a Windows azonnal átáll a kiválasztott replika használatára. A Check Status gombra kattintva pedig ellenőrizhetjük a megosztott mappák állapotát.

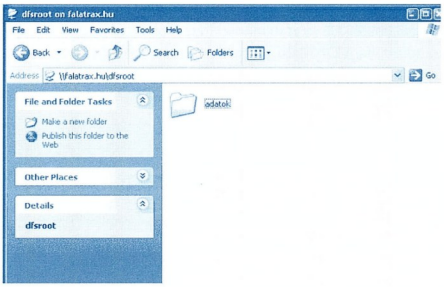
A DFS névtér nem tartalmaz az általa mutatott megosztott mappákra vonatkozó jogosultsági információkat. A DFS fa elemei minden felhasználó számára láthatók, még akkor is, ha egy adott megosztott mappához a felhasználónak nincs hozzáférési jogosultsága. Ebben az esetben a kérdéses üresen jelenik meg, tartalma nem látható.

Tartományi DFS névtér létrehozása

A tartományi DFS gyökér létrehozása szinte teljesen megegyezik az önálló DFS gyökér telepítésével. Ebben az esetben is választanunk kell egy kiszolgálót, amely a DFS gyökér kiszolgálójaként funkcionál, és meg kell adnunk a megosztani kívánt könyvtár és a megosztás nevét. A különbség mindössze annyi, hogy menet közben a varázsló segítségével választhatunk, hogy az elérhető tartományok közül melyikbe szeretnénk a gyökeret „elültetni". További érdekesség, hogy - mivel az adatok ilyenkor az Active Directory-ban tárolódnak, nem csak új DFS gyökeret hozhatunk létre, hanem egy már meglévőhöz is csatlakozhatunk, ha a „New DFS Root..." helyett a „Display an existing DFS root..." parancsot választjuk. Ha a DFS gyökérre jobb gombbal kattintunk, a „New Root replica..." paranccsal pedig nemcsak a DFS linkekhez, hanem a DFS gyökérhez is több kiszolgálót rendelhetünk, így az egyik kiszolgáló kiesésével a DFS még továbbra is használható marad. *(Sőt, tulajdonképpen a konfigurációs információ még akkor is a címtárban marad, ha véletlenül mindegyik DFS kiszolgáló kiesne).*

A tartományi DFS névtérre nem a kiszolgáló, hanem a tartomány *(DNS vagy NetBIOS)* nevével hivatkozhatunk:

```
\\falatrax.hu\dfsroot
vagy
\\FALATRAX\dfsroot
```

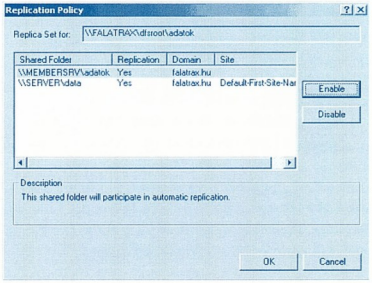


♣ *A tartományi DFS gyökér Windows XP-ben*

Tartományi DFS gyökérhez a tartomány nevével csak a Windows 2000 vagy újabb ügyfelek képesek csatlakozni, de a gyökér replikáihoz *(kiszolgálónévvel, hasonlóan, mint az önálló DFS esetén)* a régebbi Windows ügyfelek is kapcsolódhatnak. Ne felejtsük el, hogy külön kiszolgálókon bár, de egy tartományba több tartományi DFS gyökeret is felvehetünk!

A DFS replikációs házirend

Már a replikák létrehozásánál feltűnhetett, hogy tartományi üzemmódban a manuális mellett az automatikus replikációt is választhatjuk. Ha pedig a fában a DFS linkre jobb gombbal kattintunk, és a megjelenő menüben a Replication Policy... parancsot választjuk, megjelenik az automatikus replikáció beállítására szolgáló ablak:



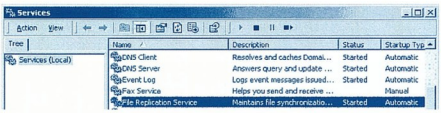
♣ *Az automatikus replikáció beállítása*

A replikációt olyan megosztások között tudjuk engedélyezni, amelyek Windows 2000 Serveren találhatók, közös, vagy egymásban bízó tartományokba tartozó kiszolgálókon. Nem feltétel, hogy a kiszolgálók egyben tartományvezérlőként is funkcionáljanak, az viszont igen, hogy a megosztott mappák könyvtárai NTFS fájlrendszeren legyenek. Hogy miért? Itt lép a képbe a másik főszereplőnk, a File Replication Service.

A File Replication Service (FRS)

A Windows tartományvezérlők rendszermappáinak megosztásait már régóta külön erre a célra készült rendszerszolgáltatások szinkronizálgatják. A Windows 2000 fájlreplikációs szolgáltatása *(FRS)* azonban nem csak a tartományi adatbázis *(azaz a SYSVOL könyvtár)* replikálására használható, hanem a DFS mappák replikái közötti szinkronizálásra is.

A File Replication Service minden Windows 2000 Server-ben megtalálható, de automatikusan csak a tartományvezérlőkön indul el. Ha a DFS-ben engedélyezzük a replikációt egy nem-tartományvezérlő Windows 2000 Server felé, az automatikusan átállítja a szolgáltatás indítási paramétereit és el is indítja azt.



♣ *A fájlreplikációs szolgáltatás*

Az FRS indítása után saját eseménynapló-fájlt hoz létre magának, File Replication Service néven.

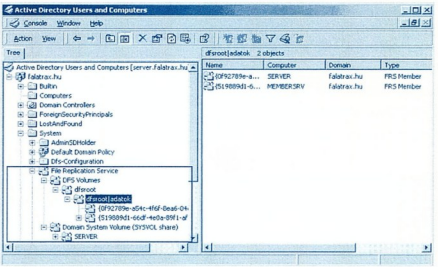
Az FRS működése

Az FRS az Active Directory-replikációhoz nagyon hasonló replikációs szolgáltatást végez. A rokonság olyan szoros, hogy többek között az FRS is figyelembe veszi az Active Directory telephelyek *(site-ok)* beállításait, a replikációs linkek adatait, és a címtár szinkronizációjával egyidejűleg működik. Ez azt is jelenti, hogy ha a szinkronizálandó megosztott mappák két különböző Active Directory-telephelyhez tartozó kiszolgálón találhatók, akkor a fájlok replikációja a telephely-kapcsolattól függő késlekedést szenved *(azaz, akár az is előfordulhat, hogy a mappák tartalma például naponta egyszer szinkronizálódik)*. Míg az Active Directory replikáció telephelyek között tömörített, az FRS az adatokat még a telephelyek közötti replikáció során sem tömöríti.

Az FRS egyébként mindig egész fájlokat szinkronizál, tehát nincs szó az adatok részleges replikációjáról. Stratégiailag pedig az „utolsó mentés érvényes" elvet követi, azaz függetlenül a mentés helyétől, az a fájl lesz végül a győztes, aminek dátumbélyege a legfrissebb az összes többi között. *(A valódi működés ennél kicsit bonyolultabb, mert az FRS a fájlokra is használ egy verziószámot, ami hasonlít az Active Directory objektumok USN-jére... részletes információkat a Resource Kitből tudhatunk meg).*

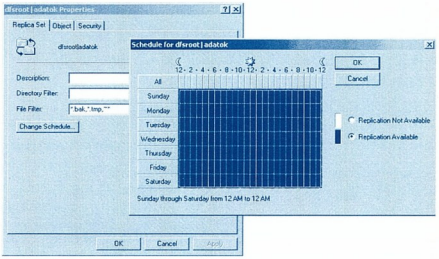
Az FRS replikációs paramétereinek módosítása

A DFS konzolban található egyszerű dialógus *(replikáljunk-e vagy sem)* mellett az FRS finomabb beállításait is elérhetjük grafikus felületen keresztül. Ehhez a tartományvezérlőn az Active Directory Users and Computers válasszuk a View / Advanced Features nézetet, majd bontsuk ki a fát a System / File Replication Service szintig!



♣ *FRS replikációs információk a címtárban*

Ez itt a DFS konfiguráció: megtaláljuk az aktuális DFS gyökerek és a bennük található DFS linkek, sőt, még a replikák listáját is. És hogy ez miért jó nekünk? Kattintsunk jobb gombbal a kívánt DFS link *(esetünkben a „dfsroot|adatok")* sorra, és válasszuk a Properties parancsot!



♣ *Az FRS finomhangolása az Active Directoryban*

A megjelenő dialógusablakban például meghatározhatjuk, hogy a mappák között milyen fájlokat és könyvtárakat nem kell replikálni *(alapértelmezésként nem foglalkozunk például a *.bak, *.tmp, és a ~ jellel kezdődő - általában átmeneti célra létrehozott - fájlokkal)*. A Change Schedule gombra kattintva pedig megjelenik az ábrán is látható időzítőablak, aminek segítségével akár DFS hivatkozásonként meghatározhatjuk a replikációs időszakokat.

Fülöp Miklós
mick@netacademia.net

Ki mivel ♥?

Farkasokkal táncoló ráadás...

Ha elkezdünk valamit, akkor azt tegyük tisztességesen - mindig ezt vallottam. Készüljünk fel a legjobb tudásunk szerint, és csak azután vágjunk bele a dologba. Akik követik a „Farkasokkal táncoló" cikksorozatot, azok tudják, hogy igyekszem alaposan körbejárni egy témát, mielőtt valóban cselekednék. Nos, az alábbi történet arról szól, hogy ez nem mindig elég.

...és legyen biztonság!

A vállalatunknál 1997 februárja óta a Windows NT 4.0 a munkaállomások standard operációs rendszere. Ismerve a szoftver megjelenési dátumát *(1996. november)*, meglehetősen bátor tett volt a szabvány bevezetése. Nem is jutott érvényre rögtön. Kompatibilitási okokból még jó ideig találkoztunk Windows 95 rendszerekkel is, és nem kevés hadakozásban vettünk részt, amikor egy-egy szállító a lehető legnagyobb természetességgel hozott Win9x rendszert, és nyafogott, ha ez nekünk nem tetszett. *(Nyafogásnak azt nevezem, ha valaki tesztek nélkül esküszik arra, hogy NT4 alatt nem fut valami, ezért Win9x-re van szükség.)* Lényeg a lényeg: a Windows NT alap lehetőséget ad egy biztonságosabb rendszer kialakítására. Tudjuk, mert ennek a magazinnak a hasábjain is bőven olvashattunk róla, a Windows 2000 egy sor újdonságot hozott a biztonság területén. Alapértelmezett hitelesítési protokollja a Kerberos lett, a tartományokat csoportházirendek és biztonsági sablonok segítségével lehet gyorsan, szabványosan felügyelni. Igaz, a csoportházirendek csak Windows 2000 vagy magasabb verziójú kiszolgálók és ügyfelek esetén jutnak érvényre, de sebaj, ha tisztán W2K hálózatunk van *(vagy lesz)*, kisebb fajta Kánaánba juthatunk, - már ami a felügyelhetőséget illeti.

Az ISA log

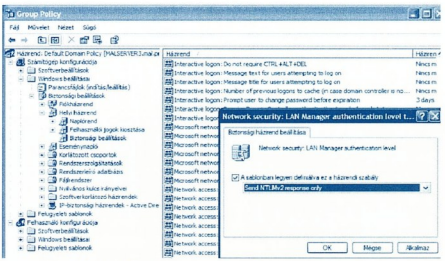
Történt aztán, hogy ISA szerverre cseréltük a tűzfalunkat, megújítva a korábbi MS Proxy 2.0-át. Az ISA egyik új szolgáltatása a jelentések készítése. Kissé korlátozottak a képességei az alapterméknek, mégis előrelépés a Proxyhoz képest.

Az ISA telepítése után egy-két nappal nézegettem a kimutatásokat, és a használt operációs rendszerek megoszlásakor csodálkozva fedeztem fel, hogy bizony Win9x rendszerek is vígan töltögetik le a maguk kis weblapjait. Ez nekem nem tetszett. Öt évvel egy szabvány bevezetése után még létezik olyan eldugott sarka a hálózatunknak, ahol Windows 95 garázdálkodik? Vagy kalózgépek kerültek a céghez? Nem hagytam annyiban a dolgot, és úgy határoztam, hogy lehetetlenné teszem a Win9x rendszerek életét a vállalatnál. Hogyan lehetséges ez? Nem is olyan nehéz, mindjárt kiderül.

Legyen *(nagyobb)* biztonság!

Aki ismeri a két rendszer felépítését, az tudja, hogy alapvetően a biztonság kérdésköre az, ahol az NT-alapú rendszerek lekörözik a Win9x vonalat. Felhasználók, privilégium-szintek, jogosultságok, hitelesítés, ha egyáltalán léteznek, mind-mind nyögvenyelős funkció a hagyományos Windows világban. Nem vállalati környezetbe tervezték a terméket, no. Ha van tehát gyenge pont, ahol meg lehet távolról és ismeretlenül fogni egy Win9x gépet, akkor ez a biztonság lesz. És valóban: egy egyszerűen telepített Windows 95/98 nem ismeri például az NTLMv2 hitelesítést. Igaz, a Windows 2000-hez adott Active Directory kliens segédprogram telepítésével ez a korlát már nem korlát, de joggal feltételeztem, hogy a Windows 95 rendszer használója nem ismeri az így keletkező járulékos előnyöket, és az AD klienst nem telepítette.

Az ötlet egyszerű: tiltsuk le csoportházirend segítségével a LANMAN és az NTLM hitelesítést. Mi értelme van ennek a beállításnak, ha az ügyfelek 99%-a NT4 munkaállomás? Látszólag nem sok. Csakhogy: mi végzi el a hitelesítéseket? A tartományvezérlő. Abból viszont nálunk már csak Windows 2000-es létezik. Nosza, adjuk meg tartományszinten azt a szabályt, hogy DC-k csak NTLMv2 hitelesítést fogadjanak el. Okozhat ez problémát? Az NT4-es gépeknek nem, ha van rajtuk 4-es javítócsomag, azt pedig még 1999 decemberéig *(a dátumváltási mizéria idején)* mindegyik megkapta. Sőt, a gépek 90%-a SP6-os csomaggal fut. A Windows 2000-es szervereknek sem lesz semmi bajuk, hiszen egymás között Kerberost használnak, az NTLMv2 pedig a kisujjukban van. Egyedül a kósza Win9x-es gépeknek lesz problémájuk. Épp ezt szeretném: telefonáljon nekem az a felhasználó, akinek gondja van.



♣ ***Egy beállítás - kicsit pontatlanul***

Beállítottam, amit be kell, és vártam. Ha ránéz a képre az Olvasó, rögtön láthatja, hogy a dolog egy picit sántít. Ez a beállítás arról szól, hogy a szerverek *(hiszen csak rájuk érvényes még a házirend, mert csak ők tudják alkalmazni)* csak NTLMv2 csomagot küldenek ami azonban nem zárja ki, hogy LM vagy NTLM csomagot fogadjanak. Ez egy tévedés volt a részemről, és a tényleges kísérletemet meghiúsította volna, de a történetünk szempontjából ez most kevésbé érdekes.

Sokkal fontosabb az, hogy nem történt semmi. Eltelt egy-két nap, és egyetlen telefonos bejelentés sem érkezett, amelyből azt lehetett volna kihámozni, hogy itt hitelesítési problémába ütközött a felhasználó.

A dolog annyiban maradt. A beállítást, - lévén, hogy ártalmatlan - nem piszkáltam a továbbiakban.

Kell neked biztonság? Nesze!

Ahogy a mesében, eltelt egy hét, eltelt kettő... sőt lehet, hogy tizenkettő. Az egész dologról megfeledkeztem. Egyik nap látom ám, hogy a vállalatirányítási rendszerünk cluster kiszolgálói közül az egyiken nem indul a fürtszolgáltatás. Sajnos nincs szép képem, amelyet megmutathatnék, de az esemény pontos leírásával szolgálhatok.

```
Event ID 9

The device, \Device\Scsi\Cpqfcalm2 did not respond
within the timeout period.
```

Szép, csupa piros eseménynaplónk volt. A Cpqfcalm2 megnevezés a host bus adatpereket rejti, amelyek a lemez-alrendszerrel biztosítják a kapcsolatot. Mivel az adapterek nem válaszoltak, nincsenek lemezek. Ha pedig nincsenek lemezek, akkor nem csoda, hogy a fürtszolgáltatás nem indul.

```
Event ID 1009

The Clustering Service could not join an existing
cluster and could not form a new cluster. The
Clustering Service has terminated.

Event ID 7031

The Cluster Service service terminated unexpected-
ly. It has done this 2 time(s). The following
corrective action will be taken in <100000> mil-
liseconds. Restart the service.
```

Ez bizony csúnya hardverhiba. Van ugyan egy kis bökkenő, nevezetesen, hogy egyszerre két HBA ment tönkre, de legalább van fürtszolgáltatásunk, és az ERP rendszerünk nem áll meg. Be kell jelenteni a szállítónak, az majd jön, kicseréli a hibás alkatrészeket, aztán helyreáll a rend és a béke. Mivel sima ügyről van szó, „végrendelkeztem" a kollégámnak, és elmentem tanulmányi szabadságra, ahogy azt már jóval korábban elterveztem.

Pár nap múlva megérkezett a segítség. A gyártó szakmérnöke megnézegette a hardver eseménynaplóit, de semmi hibát nem talált. Azért az mégiscsak furcsa, hogy két eszköz is hibás, de a diagnosztika szerint hibátlan. A szokásos „indítsuk újra", meg „próbáljuk csak a fürtszolgáltatást" ötletek után következett az „indítsuk újra a hibátlan másik fürtöt" is. Ekkor jött az igazi meglepetés: a mindaddig hibás HBA kártyák hirtelen feléledtek, „meglátták" a közös lemezeket, a fürtszolgáltatás pedig elindult. Az újraindítást követően viszont a másik állomás kezdett az elsőhöz hasonló tüneteket produkálni. Hmm.

Ekkor kaptam az első telefont, mint felkent fürtszakértő. Bámultam, mint borjú az új kapura. Olvastam én már mindenfélét, de ilyen hibáról még nem hallottam. Egyre csak azon gondolkodtam, hogy ez hardver- vagy eszközvezérlő hiba lehet, hiszen mi másról lenne szó, ha abból a rétegből jön a jelzés, és jelzi: a „HBA nem válaszol".

A kollégák azonban kevésbé voltak betokosodva. Az volt a sejtésük, hogy ez fürthiba, ha pedig fürthiba, akkor a TechNet CD, vagy a frissebb weblap segíteni fog, legalábbis ötletet ad. Nem tévedtek, mert a következőket találták: Q272129. A hiba leírása rövidítve a következő:

Vagyis: a probléma akkor jelentkezhet, ha szigorú biztonsági sablont alkalmazunk, vagy kézzel állítjuk át a hitelesítési szintet bármi másra, mint „LM és NTLM" a Windows 2000 alapú fürtökön. A fürt nem működik helyesen, ha NTLMv2-t használ. Minden hitelesítés belső eljárásokkal történik, miután a fürtszolgáltatás RPC csomagokkal létrejön. Csak a fürthöz való csatlakozáskor fordul a cluster a tartományvezérlőkhöz, hogy a cluster fiókot hitelesítse. Minden állomást, amely csatlakozni akar a fürthöz, a magánhálózaton keresztül hitelesíti a quorum tulajdonosa. Ekkor csak LM és NTLM módszert használ a kiszolgáló.

Még egy mozdulat a kollégák részéről, és megtalálták a hibás beállítást. De ki állította be? Újabb hívást kaptam. „Nem tudom, válaszoltam, de nem köszöni meg tőlem, amit kapni fog" - feleltem. Aztán rövid idő múlva derengeni kezdett a dolog, visszahívtam a fiúkat, és töredelmesen bevallottam, hogy az a valaki valószínűleg én voltam. Ez bizony égés. Ráadásul nem kevés pénz is.

Az első munkanapomon megnéztem a fürt naplóit:

```
[CS] Service Domain Account = MAL\clustuser
[CS] Initializing RPC server.
[INIT] Attempting to join cluster MAL-ENTERP1
[JOIN] Spawning thread to connect to sponsor
192.168.111.2
[JOIN] Asking 192.168.111.2 to sponsor us.
[JOIN] Unable to get join version data from spon-
sor 192.168.111.2 using NTLM package, status 1825.

[JOIN] JoinVersion data for sponsor 192.168.111.2
is invalid, status 1825.
[JOIN] All connect threads have terminated.
[JOIN] Unable to connect to any sponsor node.
[INIT] Failed to join cluster, status 53
```

Az 1825-ös hiba hitelesítőcsomaghoz kapcsolódó problémát jelöl.

Tanulságok

1. A munkánk során tele vagyunk hibás következtetésekkel. Különösen igaz ez akkor, ha egy összetett, minden részletében nem ismert alrendszerről *(GPO, cluster)* van szó. A házirend például nem érvényesült a tartományvezérlőkön, mert azokra egy hozzájuk közelebbi GPO felülírta az én beállításomat. Ez egyébként hátráltatta a problémamegoldást, mert ha több fürtön hasonló hiba jelentkezik, az ember előbb kezd központi hibaforrásra gyanakodni.
2. Lehetséges olyan globális beállítás, amely funkcióban tőle távoli rendszereket érint. Ez a szomorú valóság.
3. Nem elég sokat olvasni - még többet kell.
4. Hasznos elemezni a fürt naplóját, sokat segíthet. Ha ezt a szabadságom előtt megteszem, megtakarítottam volna egy hétvégi munkát a kollégáimnak.

Mindehhez hozzá kell tenni, hogy a sokat emlegetett biztonság kontra használhatóság vonalat le kellene cserélni a biztonság-használhatóság-kompatiblitás háromszögre, amikor a rendszereinket tervezzük. A fenti hiba ugyanis azt jelenti, hogy amíg a gyártó *(jelen esetben a Microsoft)* ki nem javítja a fürtszolgáltatás hitelesítő alrendszerét, addig együtt kell élnünk régi, kevésbé biztonságos szabványokkal. Ha van igazán szomorú következtetés, akkor azt hiszem, hogy ez az. Szép dolog a Windows 2000 sok új biztonsági újítása, de akik a magas rendelkezésre állás útját választják, azoknak egyelőre várni kell az új világra. A következő javítócsomagig? A következő verzióig? Vagy még tovább?

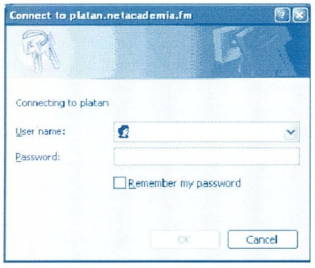
Lepenye Tamás, MCSE 2000
lepenyet@mal.hu

Windows XP: jelszóvédelem

Hiába él együtt az ember huzamosabb ideig egy Windows XP-vel, teljesen kiismerni soha nem fogja. Ha csak egy kicsit is letéved a megszokott ösvényről, máris újabb változások nyomára bukkan. Így jártam én is a jelszókezeléssel...

Connect As...

Vegyünk két Windows XP-t, melyek nem ugyanannak a tartománynak a tagjai. *(Kössük össze például gondolatban az én laptopomat a tiéddel.)* Ha saját gépünkön bejelentkezünk a saját nevünkben, természetesen hozzáférünk saját adatainkhoz. Ha azonban a másik gép megosztott erőforrására szeretnénk csatlakozni, feljön a szokásos „Connect As...", vagy „bejelentkezés más néven" ablak, amit megfelelően ki kell töltenünk a sikeres csatlakozás érdekében - vagyis egy olyan névjelszó párost kell megadnunk, mely a Te gépeden érvényes. Emlékeztetőül, erről az ablakról beszélek:



Átjelentkezés egy másik számítógépre, más felhasználó nevében

Ha gyakran szükség lenne erre a kapcsolatra, érdemes lenne elmenteni a név-jelszó párost a „Remember..." pipa bejelölésével. Hopp! Windows 2000-ben nem létezett ilyen pipa! Hálózati meghajtók betűhöz rendelésénél *(map)* találunk ugyan pipát, de ezt: „Reconnect at logon" *(bejelentkezéskor újracsatlakoztatás)*!

A két dolog nem ugyanaz, noha mindkettő mögött a név-jelszó páros lementése húzódik meg. *(Egyébként a profilba kerülnek ezek az adatok. Hogy milyen titkosítással? Az titok!)* Míg azonban Windows 2000 esetén semmilyen eszközt nem kaptunk a mentett jelszavak kezelésére és/vagy törlésére, a Windows XP lehetővé teszi ezen kényes, személyes adatok kezelését!

Felhasználókezelés XP-vel

A jó öreg, egységes Felhasználókezelőtől *(User Manager)* az NT4-gyel együtt elbúcsúztunk. A Windows 2000 Professional egy MMC modult tartalmaz ugyanezen feladatok elvégzésére, melynek neve: Helyi felhasználók és csoportok *(Local Users and Groups)*. *(Fellelési helye: Sajátgép, jobbklikk->Kezelés. A bal oldali fában látható.)* A Windows XP újdonságainak kezelésére azonban ezt a jószágot nem készítették fel, sok feladat kizárólag a Vezérlőpult megfelelő ikonjával végezhető el. Ha a „Felhasználói fiókok" ikon mögött kiválasztunk egy felhasználót, lehetővé válik nemcsak a piktogram megváltoztatása, hanem egyéb, értelmes feladatok elvégzése is.

A felhasználókezelés „mélységei": piktogramot választunk!

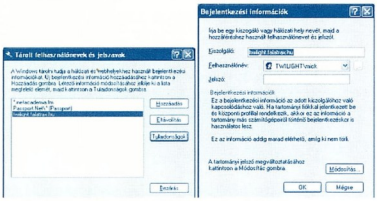
A képcserét kizárólag annak illusztrálására hoztam fel, hogy vannak olyan feladatok, amelyekkel az MMC modul nem tud megbirkózni. Vannak bizony értelmes feladatok is. A következő ábra bal felső téglalapjában, a „Kapcsolódó feladatok" között találjuk például a hálózati jelszavak kezelését!

A fiókokkal végezhető feladatok között találjuk a hálózati jelszavak kezelését és a gondűző flopilemez készítését

Hálózati jelszavak kezelése

A jelszókezelő ablakkal saját, korábban megadott *(és elmentett)* jelszavaink karbantartására nyílik mód. Kattintsunk rá, és megtekinthetjük, hányféle jelszót mentett el a rendszer eddigi futása során! Én például meglepetve tapasztaltam, hogy egyik gépünk azért nem kér tőlem sohasem jelszót, mert Fülöp Miki valaha egyszer az életben az én bejelentkezett munkamenetemből arra csámborgott, és el is mentette ***saját*** jelszavát!

Tudtál róla, mICK?



♣ *Elmentett jelszavak különböző hálózati erőforrásokhoz történő csatlakozás céljára*

Szintén megfigyelhető, hogy az erőforrások nevében dzsókerkarakterek szerepelhetnek *(mindjárt a legelső mentett jelszónál: *.netacademia.net)*. Ezzel az ügyes húzással drámai mértékben lecsökkenthető az elmentett jelszavak száma! A „Hozzáadás" nyomógombbal tetszőleges kiszolgálóhoz tetszőleges név-jelszó párost felvehetünk.
A tökéletes élményhez már csak egyetlen dolog hiányzik: az Internet Explorer-integráció. Sajnos az Explorer mind a mai napig saját jelszótárolót üzemeltet. Mentségére legyen mondva, hogy nála a jelszavak csupán űrlapelemek, melyeket az automatikus kiegészítés részeként a többi űrlapmező értékével együtt kezel, tárol, töröl.

Elfelejtettem a jelszavam!

Ez baj, nagy baj! De ma már nemcsak hackermódszerek léteznek a probléma orvoslására, hanem a Windows XP is tartalmaz beépített búfelejtő megoldást, melynek neve: „Jelszó elfelejtésének megelőzése"!
Ennek két szintje van. Egyfelől a felhasználó jelszavának beállításakor megadható egy emlékeztető mondat, amit a feledékeny ember megtekinthet hibás bejelentkezései alatt. *(Ez a lehetőség csak a helyi felhasználói fiókokra vonatkozik egyelőre, az Active Directory nem tudja!)*

♣ *Emlékeztető mondat a feledékenység ellen!*

A másik lehetőség a búfelejtő flopi használata. Ennek működése a legromantikusabb hackertúlokéra hasonlít: csak bedugunk egy flopit a gépbe, és hipp-hopp átírjuk vele a rendszergazda jelszavát!

Előtte azonban létre kell hoznunk a csodaflopit. *(Ismét olyan vizeken evezünk, ahol Active Directory még nem járt! Csak a helyi fiókok jelszavának helyreállítása lehetséges ilyen módon!)*

Az elfelejtett jelszavak varázslója

Ez az eszköz arra képes, hogy még most, teljes öntudatunk birtokában készítsünk egy olyan flopilemezt, melynek segítségével későbbi, öntudatlan önmagunkon segíthetünk. A Felhasználókezelőből indítva egy-két lépés után kezünkben a „biztonságot" nyújtó flopi, melyen mindössze egy két kilobájtos, userkey.psw névre hallgató fájl található.

♣ *Jelszókiadó varázsló*

Talán meglepő ha azt mondom, ezen a lemezen nem a mi jelenlegi jelszavunk van, hanem valami egészen más. Ezt a flopit, ha lehet, nagyon dugjuk el! Ha végignézzük a felhasználás menetét, rájövünk: ennek segítségével bárki képes lesz jelszó nélkül jelszót állítani a gépünkön!

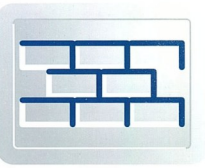
A búfelejtő flopi használata

1. Megpróbálunk bejelentkezni az általunk tudni vélt jelszóval.
2. Ha nem sikerül, Windows XP esetén ***feljön egy ablak***, ahol a „Reset password" opciót választva új jelszót adhatunk meg!
3. A jelszókiadási flopival ***azonosítjuk magunkat***, ezután tetszőleges új jelszót megadhatunk.

Vajon ezután újra el kell készítenünk a csodaflopit? Nem! Mert nincs is rajta a jelszavunk! Mit tartalmaz a lemez? Egy digitálisan aláírt adatblokkot a számítógépünkről és saját bejelentkezési nevünkről. Itt az autentikációnak egy, a hétköznapi életben gyakran, de informatikában ritkábban használt megoldásáról van szó: én vagyok én, mert ***birtoklok valamit***. Hasonlóképpen a vállalati ajtónyitogató kártyákhoz, itt is azt feltételezzük, hogy az egyedi azonosítót hordozó ***valami*** mindig a jogosult személy birtokában van. Ha nem, akkor nem. De erről a gép már nem tehet!

Fóti Marcell
marcellf@netacademia.net

Windows 2000 IP Security - 2. rész



Előző számunkban bemutattuk, hogyan hozhatunk létre szűrőlistákat és szűrőakciókat, hogy azok később az IPSec házirend építőkockáiként szerepelhessenek. Most eljött az ideje,hogy fel is használjuk őket; egyszerű IPSec házirendet hozunk létre, és bemutatjuk az alapvető adminisztrációs és diagnosztikai eszközöket is.

A teszthálózat

Cikkünkben igyekszünk mind a Windows 2000, mint a Windows XP munkaállomások üzemeltetéséhez segítséget nyújtani. A Windows XP-ben több olyan eszközt is használhatunk, ami a Windows 2000-ben még nem, vagy csak nehézkesen állt rendelkezésre. Éppen ezért „gondolatban" felépített hálózatunkban Windows 2000 és XP munkaállomások egyaránt szerepelnek.

♣ A cikkben példaként szereplő hálózat egy tartományvezérlőből valamint két munkaállomásból áll - ezek közül egyik nem tagja a tartománynak

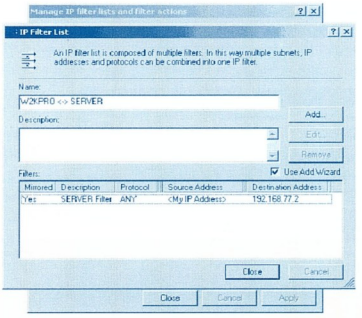
További különbség még az is, hogy az XP-t futtató számítógép nem tagja annak a tartománynak, aminek viszont a Windows 2000 munkaállomás - és tartományvezérlőként persze a Server is - igen. Ha fellapozzuk az előző számot, látni fogjuk, hogy ez a tény a háromféle azonosítási mód - tanúsítvány, szöveges, Kerberos - közül az utóbbit eleve kizárja majd.

Szűrőlisták és -akciók a munkaállomásokon

Miután mai témánk inkább az IPSec házirendek létrehozására, valamint az egész rendszer működésére összpontosít, egyelőre nagyon egyszerű szűrőlistákat és akciókat definiálunk: „minden kommunikáció, ami a munkaállomások és a kiszolgáló között zajlik, legyen titkosított". Mint tudjuk, tartományi környezetben a beállításokhoz használhatnánk a csoportos házirendet is, de az egyszerűség kedvéért használjuk továbbra is a helyi biztonsági házirendet a szabályok létrehozásához.

Hozzunk létre a munkaállomásokon egy-egy szűrőlistát, „W2KPRO <-> SERVER" illetve „WXPPRO <-> SERVER" néven, és mindegyikhez vegyünk fel egy-egy *szűrőt*, az alábbi beállításokkal *(a Server IP címe 192.168.77.2)*:

```
Source Address: My IP Address
Destination Address: 192.168.77.2
Protocol Type: Any
Mirrored: Yes
Description: Server Filter
```



♣ A munkaállomások szűrőlistája most nagyon egyszerű

Hasonlóképpen hozzuk létre a munkaállomásokon szűrőakciót, az alábbi beállításokkal:

```
Name: My Filter Action
General Options: Negotiate Security
Communicating : Do not communicate
IP Traffic Security: High (Encryption and
Integrity)
```

Ez az akciót fogjuk majd használni a számítógépek kommunikációja során.

A Default Response Rule

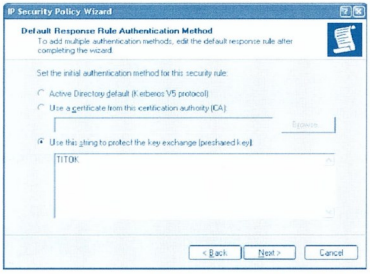
Mielőtt létrehozhatnánk az első házirendet, meg kell ismernünk egy újabb fogalmat. Az eddigiek során mindig arról beszéltünk, hogy mi történjen a számítógépről *kiinduló* csomagok elküldése során. A szűrőlisták és -akciók pontosan meghatározzák majd, hogy a számítógép hogyan viselkedjen, mielőtt egy *kimenő* kapcsolatot kezdeményez. De mi történjen, ha a kapcsolatot nem mi kezdeményezzük, hanem *(akár titkosítatlanul,*

akár titkosítva) a másik fél, és hozzánk csak a titkosítatlan csomag, vagy éppen az IPSec csatorna felépítésének kérelme érkezik meg? Ezt a reakciót határozza majd meg a számítógépen érvényben lévő házirend egyik szabálya, amit Default Response Rule-nak hívnak. A házirend, mint tudjuk, szabályok halmaza lesz, és e szabályok közül ez lesz az, ami a beérkező kapcsolatok kezelését irányítja. Ha a házirend nem tartalmaz Default Response Rule-t, a beérkező IPSec csatorna építési kérelmekre a számítógép nem fog válaszolni *(hiszen szabály hiányában nem tudja, mit és hogyan engedhet meg magának)*.

Az IPSec házirend létrehozása

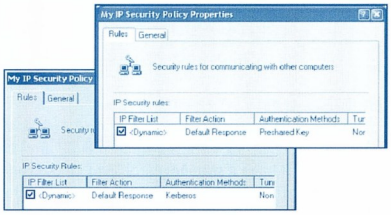
A házirend létrehozásához kattintsunk jobb gombbal az IP Security policies MMC modul felületére, és válasszuk a Create IP Security Policy… menüpontot. Ezzel új, saját házirendet hozunk létre a gyárilag definiált három mellett. Haladjunk végig a varázsló által felkínált lépéseken:

- Adjunk nevet a házirendnek: például My IP Security Policy
- A varázsló következő kérdése arra vonatkozik, hogy aktiváljuk-e a Default Response Rule szabályt. Ezt a fentiek ismeretében természetesen engedélyezzük, a szabály részletes tartalmára később még visszatérünk.
- A beérkező kérésekkel kapcsolatban még egy dologban kell nyilatkoznunk: milyen számítógép-azonosítási módszerrel reagáljon a beérkező kérésekre. Itt a Windows 2000 Professional esetén válasszuk a Kerberost, a Windows XP-ben pedig a szöveges *(Preshared Key)* opciót, és válasszunk is egy jelszót; legyen ez „TITOK”.



♣ A szöveges azonosítás jelszava nyíltan látszik a házirendben (és többek között a registryben is)

A varázsló ezután befejezi a működését, és megjeleníti nekünk a frissen létrehozott házirend tulajdonságlapján, ami jelenleg egyetlen szabályt tartalmaz.



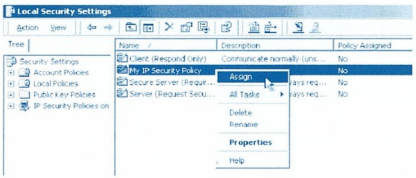
♣ Az új házirendben csak a Default Response Rule szabály található

Miután a bejövő kapcsolatok sorsát nagyjából eldöntöttük, hozzuk létre azt a szabályt, ami a kimenő kommunikációt határozza meg! Kattintsunk a házirend tulajdonságlapján az Add.. gombra, és máris újabb varázsló indul: ez a Security Rule Wizard. Feladatunk most tehát az, hogy az első körben létrehozott szűrőlistát és -akciót hozzárendeljük a házirendhez.

- Nem építünk IPSec alagúthálózatot *(arra van más megoldás a Windowsban)*, ezért az első oldalon válasszuk az alapértelmezett „This rule does not specify a tunnel” opciót
- A következő oldalon kiválaszthatjuk, hogy a szabály csak a RAS, csak a helyi hálózati vagy minden hálózati kapcsolatra érvényesüljön *(válasszuk ismét az alapértelmezést: All network connections)*
- A következő lépésben ismét a számítógép-azonosítási módról kell nyilatkoznunk. Itt, a bejövő kapcsolatokkal hasonlóan, ismét válasszuk a Kerberost *(W2K)* illetve a szöveges *(WXP)* azonosítást, és adjuk meg a jelszót is: „TITOK”
- Ezután végre kiválaszthatjuk azt a szűrőlistát, amire a szabály vonatkozni fog. Válasszuk ki a „W2KPRO <-> SERVER” illetve „WXPPRO <-> SERVER” szűrőlistákat
- Most a szűrőakció kiválasztása következik: itt jelöljük ki a „My Filter Action”-t

A varázsló kilép, és a házirendben a Default Response Rule mellett máris megjelenik az újonnan definiált szabály. A szabályok melletti jelölőnégyzetekben látható pipa jelzi, hogy aktív házirend esetén a szabály érvényben van.

Zárjuk be a házirend tulajdonságlapját és érvényesítsük a házirendet! Ehhez jobb gombbal kattintsunk a listában a My IP Security Policy sorra, és a menüből válasszuk ki az Assign parancsot. A házirend beállításai azonnal érvényre jutnak.



♣ A házirend érvényesítése már csak egy kattintás

Házirend létrehozása a kiszolgálón

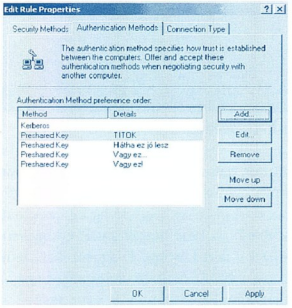
Ez azonban még kevés az üdvösséghez. Ha most megpróbálnánk a munkaállomásokról felvenni a kapcsolatot a kiszolgálóval, nem sikerülne - hiszen a Server a munkaállomásokon történt változásokról mit sem tud! *(Nem jönne létre titkosított csatorna, a munkaállomásokon kikényszerített akció pedig tiltja a titkosítatlan kommunikációt.)* Létre kell tehát hoznunk egy házirendet a kiszolgálón is. Esetünkben - mivel feltesszük, hogy a kapcsolatot mindig a munkaállomások kezdeményezik - elég, ha a házirend egy megfelelően testreszabott Default Response Rule-t tartalmaz. Itt az alkalom, hogy részleteiben is megismerkedjünk ezzel a szabállyal, ehhez azonban először létre kell hoznunk egy teljes házirendet.

- Indítsuk el a kiszolgálón a Local Security Policy eszközt, a fában kattintsunk jobb gombbal az IP Security Policies… sorra, és válasszuk ki a Create IP Security Policy utasítást
- Adjunk nevet a házirendnek, legyen például „My Response Policy”
- Válasszuk az Activate the Default Response Rule… opciót, azonosítási módként pedig egyelőre jelöljük ki a Kerberost

Ha a házirendet ebben a formájában aktiválnánk a kiszolgálón, a Windows 2000 Professional már elérné azt - hiszen a Kerberos azonosítási módot a kiszolgálón is engedélyeztük. De mi a helyzet a Windows XP-vel? A Servernek fogalma sincs arról, hogy mi a jelszó, sőt, eszébe sem jut, hogy egyáltalán szöveges azonosítást használjon! Ehhez további beállításokra van szükség.

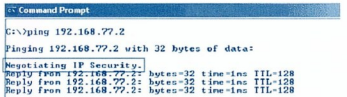
Ha az aktiváláshoz bezártuk volna, nyissuk meg a házirend tulajdonságlapját, ott kattintsunk kétszer a Default Response Rule-ra, vagy válasszuk ki azt, és nyomjuk meg az Edit... gombot. Ekkor megjelenik a szabály tulajdonságlapja - válasszuk először az Authentication Methods oldalt.

Itt láthatjuk, hogy egy szabályhoz *(és ez nem csak a Default Response Rule-ra igaz)* több azonosítási módot is hozzárendelhetünk - sőt, egy-egy azonosítási mód többször is szerepelhet a listában.



♣ *Az azonosítási módok sorrendje is módosítható*

A Move up/down... gombokkal befolyásolhatjuk az azonosítási módok sorrendjét. Ha - ahogy az ábrán is látható - a Kerberos mellé *(mögé)* felvesszük a szöveges azonosítást, a megfelelő jelszóval *(„TITOK")*, a kiszolgáló a sikertelen Kerberos próbálkozás után *(vagy annak híján)* megpróbálkozik majd a TITOKkal is. Ahogy ezt a beállítást a kiszolgálón érvényesítjük, a Windows XP is képes lesz végre a titkosított csatorna felépítésére - készen vagyunk! Próbáljuk ki, mit sikerült összehozni: például pingeljük meg a kiszolgálót mindkét munkaállomásról.

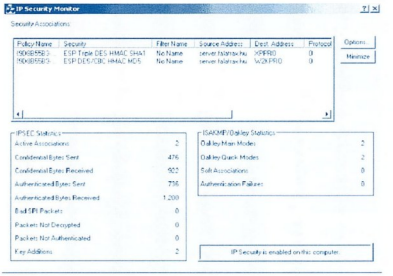


♣ *Ha még nincs aktív csatorna, annak felépülésére bizony néha kicsit várni kell*

Némi várakozás után *(„Negotiating IP Security...")* a parancs sikeresen végrehajtódik - a frissen felépített titkosított csatornán keresztül.

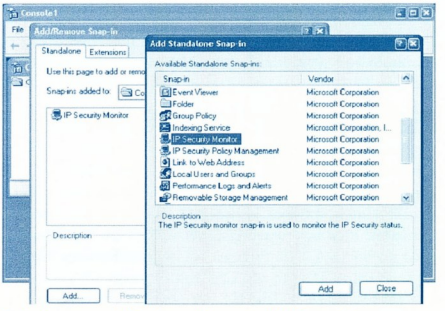
Az IP Security Monitor - először

A legegyszerűbb eszköz az IPSec csatornák működésének ellenőrzésére az IP Security Monitor. Ezt Windows 2000-ben a Start menü Run... ablakába bepötyögött „ipsecmon" parancscsal indíthatjuk el:



♣ *Az ipsecmon a Windows 2000 kiszolgálón - jól látható a két aktív IPSec csatorna*

A képernyőkép a Windows 2000 kiszolgálóról készült, közvetlenül azután, hogy a munkaállomásokról megpingeltük őt. Az ábrán kivehető, hogy jelenleg két csatorna aktív, az első a Windows XP-vel; Triple DES / SHA1 titkosítással, a másik pedig a W2K Professionallal, de itt „csak" DES / MD5 a titkosítás. Ennek az az oka, hogy a „High" titkosítási akció Windows 2000-ren a DES-t tekinti alapértelmezésnek - holott a Service Pack 1 telepítése után már az erősebb titkosítás is rendelkezésre áll. Ha van kedvünk, természetesen „kézzel" nyugodtan módosíthatjuk az akció beállításait. Az ablak alján látható statisztikai adatokra most nem térünk ki, azokhoz ugyanis az IPSec protokoll mélyebb ismeretére lenne szükség - de ami késik, nem múlik, következő számunkban visszatérünk erre. Windows XP-ben az IP Security Monitort már az MMC modulok között kell keresnünk. A Start / Run... ablakból indítsuk el az mmc.exe-t, majd a File menüből válasszuk az Add/Remove Snap-in parancsot.

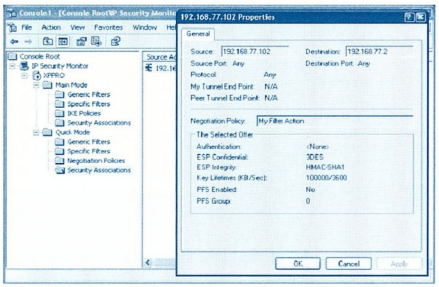


♣ *Az ipsecmon Windows XP-n már MMC modul*

A beépülő modulok listájából válasszuk ki az IP Security Monitort, az Add gombbal vegyük fel azt a telepített modulok listájába, majd a Close gombok segítségével térjünk vissza az MMC főablakába. Ezt a műveletsort legközelebb már megspórolhatjuk, ha a File menü Save As... parancsával a fris-

sen létrehozott felügyeleti eszközt elmentjük az Administrative Tools menübe.

Miután ezzel megvagyunk, itt is belekukkanthatunk a pillanatnyilag érvényes IPSec megállapodások *(SA-k)* listájába. A Quick Mode / Security Associations alatt láthatjuk az éppen aktív csatornákat, egy-egy bejegyzésen duplán kattintva láthatjuk az aktuális SA paramétereit.



♣ ***A Windows XP IP Security Monitora - rengeteg részletes információval***

Ugyanitt a Generic Filter listában megtaláljuk az érvényben lévő szűrőket *(a Specific Filters ugyanezeket tartalmazza, de „kibontva": itt már látszik, ha egy szűrő tükrözött; illetve az is, ha a számítógépnek több IP címe van - ilyenkor ugyanis minden egyes IP címhez automatikusan jönnek létre újabb és újabb szűrők)*. A Negotiation Policies alatt az érvényes szűrőakciókat látjuk; itt szerepel majd az általunk létrehozott „My Filter Action" - de van itt egy másik is, nem túl sokat mondó „1" névvel. Ha viszont belekukkantunk, máris láthatjuk, hogy ez bizony a Default Response Rule akciója. Ha a számítógép nevére jobb gombbal kattintunk, a Statistics... menüpont segítségével előcsalogathatjuk az IPSec statiszikáit. A részletekre itt is egy későbbi számunkban térünk majd ki.

Karakteres diagnosztikai információk

Ha telepítjük a Windows 2000/XP Support Tools eszközcsomagot (minden Windows CD-n megtalálható a \Support\Tools alkönyvtárban; XP-n figyeljünk, hogy „Complete" módban telepítsük), a netdiag parancs segítségével kilistázhatjuk az érvényes IPSec házirend beállításait.

```
netdiag /debug /test:ipsec
```

A válaszként kapott szöveghalmazban megtaláljuk az aktív IPSec házirend nevét, az érvényes szűrők listáját, azok beállításait (IP címek, portok), a felkínált titkosítási (OFFERS) és azonosítási módokat is (AUTHENTICATION INFO). Az alábbi példa a Windows 2000 Professional munkaállomáson készült:

```
Local IPSec Policy Active: 'My IP Security Policy'
IP Security Policy Path: {. . .}

There are 2 filters

SERVER Filter
 Filter Id: {. . .}
 Policy Id: {. . .}
 IPSEC_POLICY PolicyId = {. . .}
 Flags: 0x0
 Tunnel Addr: 0.0.0.0

 PHASE 2 OFFERS Count = 1
   Offer #0:
     ESP[ DES MD5 HMAC]Rekey: 0 seconds / 0
bytes.
 AUTHENTICATION INFO Count = 1Method = Kerberos
   Src Addr : 192.168.77.101
 Src Mask : 255.255.255.255
 Dest Addr : 192.168.77.2
 Dest Mask : 255.255.255.255
 Tunnel Addr : 0.0.0.0
 Src Port : 0 Dest Port : 0
 Protocol : 0 TunnelFilter: No
 Flags : Outbound

SERVER Filter - Mirror
...
```

♣ ***A netdiag parancs kimenetének részlete***

A *netdiag /debug /test:ipsec* parancs Windows XP-n ennél jóval szűkszavúbb, ott ugyanis a részletek listázására külön eszköz van *(ugyancsak a Support Tools része)*, ez pedig az ipseccmd. Ezzel a paranccsal „kívülről" is kezelhetjük az IPSec házirendeket, erre is visszatérünk majd később. Diagnosztikai szempontból számunkra most inkább csak az a lényeg, hogy az eszköznek részletes diagnosztikai kimenete is van, amihez az alábbi parancs segítségével juthatunk hozzá:

```
ipseccmd show all
```

Az IPSec Policy Agent rendszerszolgáltatás

A Windows IPSec szolgáltatásának lelke az IPSec Policy Agent rendszerszolgáltatás. Ez az a programmodul, ami - ha fut - az IP stacken áthaladó hálózati forgalmat szükség esetén feltartóztatja, kiépíti az IPSec csatornát, majd azon továbbítja azokat. Ugyanez a szolgáltatás felelős a beérkező forgalom kezeléséért, a csatornák megújításáért, a teljes kulcsmenedzsmentért - szinte mindenért. Ha szeretnénk, hogy a módosított beállításaink azonnal és automatikusan érvényre jussanak; ha azt akarjuk, hogy a felépített csatornák bomoljanak le, hogy azután a Windows az új paraméterek alapján újabbakat építsen, állítsuk le, majd indítsuk újra az IPSec Policy Agent szolgáltatást:

```
C:\>net stop policyagent
C:\>net start policyagent
```

Vigyázat, a parancs hatására minden IPSec csatorna megszűnik, így minden kapcsolat, ami ezeken haladt keresztül, megszakad!

folytatjuk...

Fülöp Miklós
mick@netacademia.net

VPN - Virtuális magánhálózatok

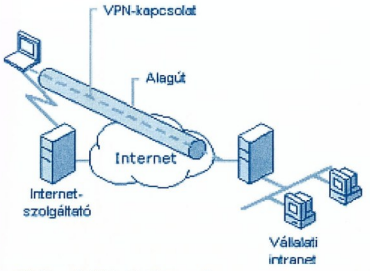
Virtuális magánhálózat: nyilvános hálózaton *(például Interneten)* keresztül megvalósított, titkosított hálózati kapcsolat, amellyel az ügyfél számítógépe vagy akár egy teljes fiókirodai hálózat hozzáférhet a központi, belső vállalati intranetre csatlakozó erőforrásokhoz.

Miért jó a VPN?

Ha néhány évvel ezelőtt egy felhasználónak otthonról *(vagy bárhonnan máshonnan, a cég területén kívülről)* el kellett érnie a vállalati erőforrásokat, szinte gondolkodás nélkül mondtuk a választ: a megoldás egy RAS kiszolgáló telepítése, ahová a dolgozó betárcsázva, a modemes kapcsolaton keresztül szépen hozzáfér mindenhez, amire szüksége lehet. A dolognak mára jó néhány szépséghibája lett:

- A RAS kiszolgálóra telepített modemek, így az egyidejűleg kiszolgálható ügyfelek száma véges
- A hívás díja akár a csillagos égig is emelkedhet, hiszen bárhol is járunk a világban, mindig a céges számot kell hívnunk - külföldről ez bizony nem olcsó mulatság
- A modemek sebessége még akkor is csak 56kbit, ha a kiszolgálóban digitális kártyákkal kapcsolódunk a telefonos hálózathoz. Ez meg sem közelíti a felhasználók számára ma rendelkezésre álló hálózati kapcsolatok *(ADSL, kábeltévé, mikró, stb.)* sebességét

A virtuális magánhálózat használata, ami nem más, mint egy, az Interneten keresztül kiépített titkosított csatorna, megoldja a problémát. Ha a cég rendelkezik egy állandó, nagy sebességű internetes kapcsolattal, a felhasználók a VPN kiszolgálón keresztül csatlakozhatnak a belső hálózatra.

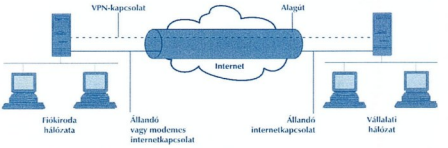


♣ *Ügyfél-kiszolgáló VPN kapcsolat*

Ekkor:

- Nincs szükség modemek használatára, a virtuális portok, azaz az egyidejű hozzáférések számát csak az erőforrások korlátozzák *(gyakorlatilag azonban csak a céges internet-kapcsolat sávszélessége jön szóba, mint korlátozó tényező)*
- A felhasználónak nem kell nemzetközi számot tárcsáznia, elég, ha bármelyik internetszolgáltatónál, helyi tarifával csatlakozik a világhálóhoz
- Modemek híján nincs sebességkorlát sem.

A VPN másik alkalmazási területe az alhálózatok összekapcsolásához kötődik. Gondoljunk egy cégre, ahol a központi hálózat mellett számos fiókiroda is működik, országszerte. Mit tehetünk, ha a fiókiroda hálózatát szeretnénk valamilyen szinten összekapcsolni a központi hálózattal? Ha megelégedtünk a modemes sebességgel, használhattuk a klasszikus telefonos megoldást. Ha azonban állandó kapcsolatra volt szükség, illetve számított a sebesség, a legjobban tettük, ha bérelt vonali kapcsolatot építettünk ki az irodák között. Mindkét megoldás méregdrága.



♣ *Kiszolgáló-kiszolgáló VPN kapcsolat*

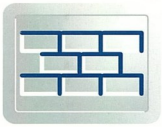
Ekkor a központi hálózat folyamatosan elérhető, a VPN kiszolgáló állandó kapcsolattal csatlakozik az Internetre. A fiókiroda internetes kapcsolata lehet állandó vagy időszaki is, az igényektől függően továbbra is megelégedhetünk a modemes kapcsolattal, vagy használhatjuk a fiókiroda nagy sávszélességű hozzáférését. A fiókiroda átjárója pedig igény szerint automatikusan felépíti a VPN kapcsolatot, és elérhetővé teszi a vállalati hálózatot - mindezt gyorsan és olcsón.

VPN protokollok

A nyilvános hálózaton keresztüli biztonságos kommunikáció érdekében a hálózati forgalmat természetesen titkosítani kell. A forgalom titkosítására többféle megoldás, protokoll létezik. A Windows 2000 ügyfelek és kiszolgálók a következő két protokoll használatát támogatják:

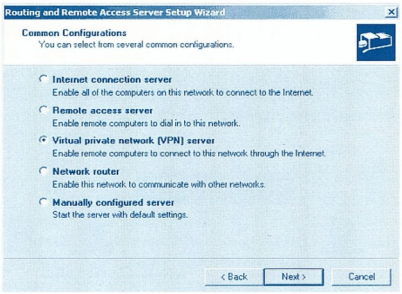
- Point-to-Point Tunneling Protocol *(PPTP)*: Az RFC 2637-ben **[1]** definiált alagútprotokoll, amely MPPE *(Microsoft Point-to-Point Encryption)* titkosítással működik. A PPTP protokollt a korábbi Windows ügyfelek *(Windows 9x-től napjainkig)* is támogatják
- Layer 2 Tunneling Protocol *(L2TP)*: Az L2TP protokoll specifikációját az RFC 2661-ben **[2]** találjuk. Maga az L2TP protokoll saját titkosítást nem tartalmaz, ezért a virtuális magánhálózatot az „L2TP over IPSec", azaz az IPSec titkosítással segített L2TP kapcsolat valósítja meg. Az L2TP használatát a Windows 2000 és Windows XP kiszolgálók illetve ügyfelek támogatják.

A protokollok részletes bemutatására később visszatérünk.



VPN kiszolgáló a Windows 2000-ben: Routing and Remote Access Service

A távelérés és útválasztás-szolgáltatás *(Routing and Remote Access Service, RRAS)* minden Windows 2000 Server beépített szolgáltatása. Telepítés után letiltva marad, mi magunk indíthatjuk el - a szükséges beállítások létrehozása után-, vagy állíthatjuk le azt, de a rendszerből eltávolítani nem lehet. Az RRAS felügyeleti konzolját az Administrative Tools menüben találjuk, nem meglepő módon Routing and Remote Access néven. Az RRAS engedélyezése előtt varázsló segít a szolgáltatás beállításában. A szolgáltatást egyébként kézzel is testre szabhatjuk, de egyelőre fogadjuk el a varázsló segítségét: készítsünk VPN kiszolgálót! Nyissuk meg az RRAS konzolt, kattintsunk jobb gombbal a kiszolgáló nevére, és válasszuk a „Configure and Enable Routing and Remote Access" parancsot!



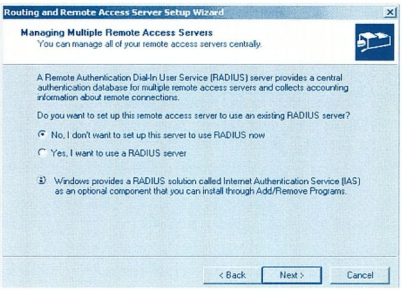
♣ ***Varázsló segít az RRAS-ból VPN kiszolgálót faragni***

A varázsló *(fenti ábrán is látható)* első oldalán válasszuk a Virtual private network *(VPN)* server opciót. Továbbhaladva ki kell választanunk a VPN ügyfelek által használni kívánt protokollokat *(ez lehet a TCP/IP mellett más is, például a NetBEUI, de a Windows kommunikációhoz ma már bőven elég a TCP/IP is)*. Az „Internet Connection" oldalon a meglévő hálózati kapcsolatok közül válasszuk ki azt, amellyel a VPN kiszolgáló az Interneten „lóg". Az RRAS ezen a kapcsolaton keresztül fogadja majd a beérkező VPN kéréseket. Ez után el kell döntenünk, hogy milyen módon szeretnénk a beérkező ügyfeleknek IP címeket osztani. Erre két lehetőségünk van:

- DHCP kiszolgáló használatával - ekkor az RRAS szolgáltatás a belső DHCP kiszolgálótól „szerez" IP-címeket.
- Meghatározott címtartományból - ekkor mi magunk határozhatjuk meg a kiosztani kívánt IP-címeket.

Bárhonnan is származik a címtartomány, annak első eleme lesz majd a virtuális kiszolgáló IP címe, a többit pedig az RRAS szépen kiosztja majd a beérkező ügyfelek között. Figyeljünk arra, hogy a kiszolgálón legyen beállítva a DNS és a WINS kiszolgálók IP címe is, mert ezeket az információkat az RRAS a bejelentkezés során továbbítja az ügyfelek irányába.

Ez után azt kell eldöntenünk, hogy milyen módon szeretnénk azonosítani a bejelentkező felhasználókat.



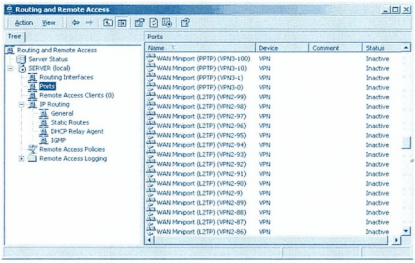
♣ ***Felhasználó-azonosítás: Windows vagy Radius?***

Ehhez a beépített Windows-módszer mellett immár a RADIUS protokoll segítségét is hívhatjuk. A RADIUS-ról később még lesz szó, egyelőre itt válasszuk a „No, I don't want to..." kezdetű sort, magyarul az azonosítást bízzuk a Windowsra. Miután ezzel elkészültünk, a varázsló elindítja az RRAS szolgáltatást. Az alapvető beállításokkal készen is vagyunk, egy híján.

Fontos! Ha az RRAS nem tartományvezérlőn, hanem tagkiszolgálón fut - ami egyébként a javasolt eljárás -, a tartományi felhasználók azonosítása csak akkor működik helyesen, ha a kiszolgálót felvesszük a „RAS and IAS Servers" csoportba. A legtöbb esetben ez automatikusan megtörténik, de mindig nézzünk utána magunk is!

Az RRAS konzol

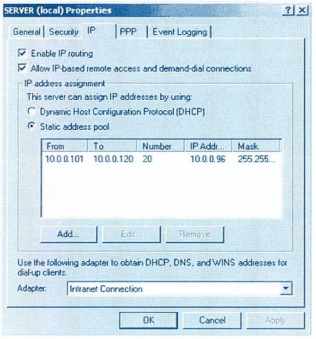
Miután a szolgáltatás elindult, ismerjük meg az RRAS felügyeleti modul tartalmát!



♣ ***Az RRAS konzol közvetlenül telepítés után***

A „Ports" sorra kattintva láthatjuk, hogy a varázsló 128-128 PPTP és L2TP portot hozott létre. Ez minden bizonnyal elég lesz, de az elérhető portok számát magunk is beállíthatjuk, ha a fában megnyitjuk a „Ports" tulajdonságlapját. Ugyanitt a portokon kettőt kattintva szükség esetén beállíthatjuk, hogy az adott eszközt szeretnénk-e bejövő, illetve kimenő kapcsolatok felépítésére használni. Az alapértelmezés természetesen nekünk teljesen megfelel. A konzolban látszik az is, ha valamelyik port éppen használatban van *(Active/Inactive)*.

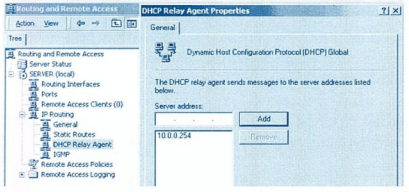
Kattintsunk jobb gombbal a kiszolgálóra *(itt SERVER (local))*, és nyissuk meg a tulajdonságlap IP oldalát!



♣ *Az RRAS kiszolgáló IP tulajdonságai*

Itt állítható be, hogy az ügyfelek IP-címe honnan származzon *(DHCP vagy meghatározott IP-cím tartomány)*. Ezt a beállítást már a varázsló megtette helyettünk. Az oldal alján kiválaszthatjuk azt a hálózati csatolót *(értelemszerűen azt, amelyik a céges hálózatra, befelé „néz")*, amin keresztül az RRAS a DHCP kiszolgálót megszólítja. Az ügyfeleknek küldött DNS és WINS címek is azok lesznek, amelyek az itt kiválasztott hálózati csatolón érvényesek!

Bár a betárcsázó ügyfél automatikusan kap IP-címet, az ő hálózatán található számítógépek is „kérhetnek" a VPN kapcsolaton keresztül. Ehhez meg kell szólítaniuk a vállalati hálózatban található DHCP kiszolgálót, ez azonban csak akkor fog sikerülni, ha az RRAS kiszolgálón futó DHCP Relay Agent segít nekik, és kérésüket továbbítja a kiszolgáló felé.

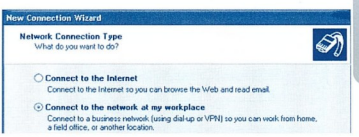


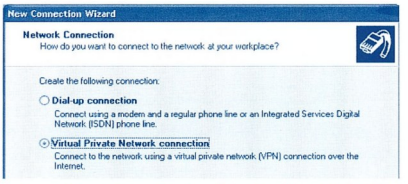
♣ *A DHCP Relay Agent továbbítja a DHCP kéréseket a kiszolgáló felé*

A DHCP Relay Agent-nek azonban meg kell magyaráznunk, hogy merre találja a DHCP kiszolgálót. Ehhez a konzolfában nyissuk meg a DHCP Relay Agent tulajdonságlapját és adjuk meg a DHCP kiszolgálók IP-címét. Emlékezzünk, hogy a betárcsázó ügyfél mindenképpen kap IP címet, a Relay Agent használatára csak akkor van szükség, ha a csatlakoztatott hálózaton belülről valaki más is szeretne IP-címhez jutni.

Ügyfél-kiszolgáló VPN kapcsolat létrehozása

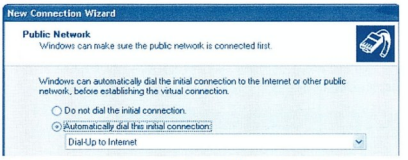
A virtuális magánhálózathoz való csatlakozásban segít nekünk az Internet Connection Wizard varázsló. Ezt a vezérlőpultból, a Network Connections ablakon belül található Create *(Make)* New Connection ikon segítségével csalogathatjuk elő.





♣ *A csatlakozástípus kiválasztása Windows XP alatt*

A csatlakozás típusánál válasszuk ki a VPN-re utaló opciót *(Windows 2000-nél „Connect to a private network...", XP-nél „Connect to the network at my workspace")*. XP esetén a következő oldalon még arról is nyilatkoznunk kell, hogy modemes vagy VPN kapcsolaton keresztül szeretnénk csatlakozni. A VPN csatorna kialakításához meglévő Internet-kapcsolatra van szükség. Ez a kapcsolat lehet állandó, de használhatunk klasszikus modemes, betárcsázós kapcsolatot is. Ezt még kényelmesebbé tehetjük, ha a VPN kapcsolatnak is beállítjuk, hogy melyik - másik - kapcsolat segítségével hozhatja létre a kapcsolatot az Internet-szolgáltatóval.

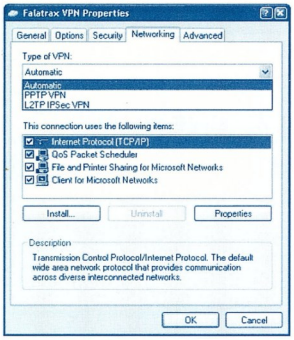


♣ *Ha kell, a Windows automatikusan tárcsázza az Internet-szolgáltatót is*

Ha ezt megadjuk, a Windows szükség esetén tárcsázza majd az Internet-szolgáltatót, és a felépült kapcsolaton keresztül fogja felépíteni majd a VPN kapcsolatot. Ezután meg kell adnunk a VPN kiszolgáló IP-címét *(mint egy „telefonszámot")*, és már készen is vagyunk.

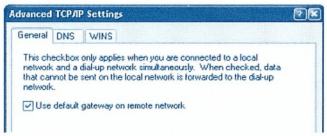
Csatlakozás a VPN kiszolgálóhoz

Ha megnyitjuk a létrehozott csatlakozási ikon tulajdonságlapját, viszontlátjuk a varázslóban megadott beállításokat. Kattintsunk a Networking oldalra!



♣ ***A VPN kapcsolat hálózati beállításai***

Látható, hogy a VPN kapcsolat típusa „Automatic". Választhatnánk kifejezetten a PPTP vagy az L2TP használatát is, de első körben érdemesebb az automatikus üzemmódnál maradni. Ilyenkor a számítógépek először megpróbálkoznak az L2TP kapcsolat kialakításával, és ha az nem sikerül, végső esélyként megpróbálkoznak a PPTP-vel.
Ha ezzel a beállítással csatlakoznánk a VPN kiszolgálóhoz, a sikeres csatlakozás után minden hálózati forgalom *(ideértve például az internetes böngészést is)* a VPN kapcsolaton keresztül igyekezne az Internet felé, ugyanis a csatlakozás pillanatában az alapértelmezett átjáró a VPN kiszolgáló lesz. Ha ezt szeretnénk elkerülni, a VPN kapcsolat tulajdonságlapjának fent is látható Networking oldalán válasszuk ki az Internet Protocol (TCP/IP)-t és kattintsunk a Properties, az erre megjelenő dialógusablakban pedig az Advanced... gombra. Ekkor a következő ablakot látjuk:

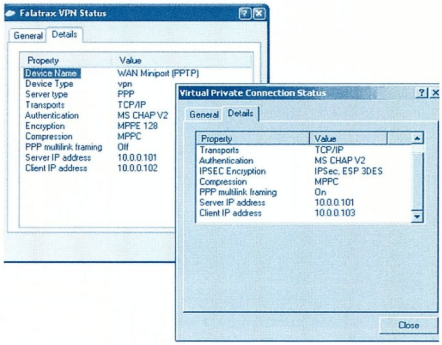


♣ ***Ha nem akarjuk, hogy minden forgalom a VPN felé menjen, töröljük ezt a beállítást!***

Ha eltüntetjük a pipát a „Use default gateway on remote network" sor elől, a következő csatlakozáskor az alapértelmezett átjáró marad az, ami volt: az Internet-szolgáltatóé. Ilyenkor a VPN kapcsolat felé csak az a forgalom halad majd, ami kifejezetten oda való. Ezt ellenőrizhetjük is a

```
route print
```

parancs kiadásával.
Miután ezzel is elkészültünk, itt az ideje, hogy csatlakozzunk végre a kiszolgálóhoz. Kattintsunk duplán a csatlakozás ikonjára vagy válasszuk a Connect parancsot! Adjunk meg egy bejelentkezésre jogosult felhasználónevet és jelszót, és kattintsunk az OK gombra. A kapcsolat néhány másodperc alatt létrejön.



♣ ***A létrejött VPN kapcsolat paraméterei***

Ha a létrejött kapcsolaton a Status parancsot választjuk, megjelenik a fent látható dialógusablak. Itt láthatjuk többek között a saját és a kiszolgáló IP-címét és a titkosítás típusát is. Ha a titkosítás típusa

- MPPE - akkor a VPN kapcsolat PPTP
- IPSec - akkor a VPN kapcsolat L2TP

protokollon keresztül jött létre. Az ábrán a bal oldali *(XP)* PPTP, míg a jobb oldali *(W2K Pro)* L2TP kapcsolatot épített fel, de ez természetesen nem az operációs rendszer típusától függ. Ne ijedjünk meg, ha elsőre nem épül fel az L2TP kapcsolat, ez az IPSec miatt még további beállításokat is igényel. Elégedjünk meg egyelőre a PPTP-vel, ami bár „gyengébb" titkosítást alkalmaz, mint az IPSec, de azért jóval több, mint a semmi.

A következő alkalommal hálózatokat kapcsolunk majd össze VPN alagúton keresztül, és nem fog kimaradni a protokollok, a PPTP, L2TP, a RADIUS és a távelérési házirendek bemutatása sem. Addig is, jó szórakozást!

folytatjuk...

Fülöp Miklós
mick@netacademia.net

A cikkben szereplő URL-ek:
[1] http://www.ietf.org/rfc/rfc2637.txt
[2] http://www.ietf.org/rfc/rfc2661.txt

Kill ME!: rendszer-visszaállítás, vírus-marasztalás

Reagálni kívánunk a III/6. számban megjelent "XP: időutazás, rendszer-visszaállítás" című cikkre, mert a bemutatott System Restore szolgáltatásnak mellékhatásai lehetnek, melyekkel a Windows ME / XP rendszereink és az ott tárolt adatok védelme érdekében érdemes megismerkedni.
A Windows System Restore ugyanis akadályozhatja a számítógépek vírusmentesítését.

Tegyük fel, hogy gépünk az I-worm.Klez **[1]** kártevő „E" változatával fertőződött. *(Mert nem telepítettünk Internet Explorer biztonsági javítást **[2]** az Outlook betekintő ablakának az ismert exploit-ok elleni védelmére és nincs, vagy nem frissítettünk víruskereső szoftvert. Esetleg a céges hálózaton egyes gépek „lyukasak", a dolgozók nem tartják be a biztonsági szabályzatot.)* A helyzet komoly: minden páratlan hónap 6-án meglepetésben lehet részünk, adatfájljaink egy része vagy az egész merevlemez random adatok halmazává válhat! Legújabb víruskergető telepítésével próbálunk véget vetni a fertőzésnek, amely detektálja a Klez-t és törli a féreg-vírus állományait. Másnap minden egyéb beavatkozás vagy látható ok nélkül ismét fertőzést jelez a vírusvédelem, a mentesítési kísérletek után a kártevő „magától" visszatér.

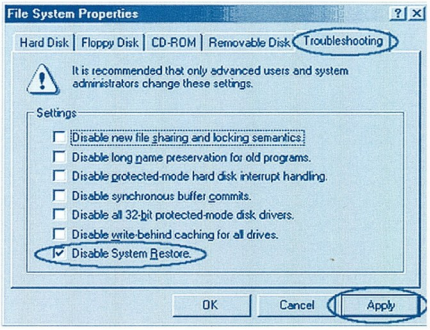
A jelenség alapvető okaként a System Restore-t nevezhetjük meg, mivel ez a szolgáltatás nemcsak a megváltozott, hanem az új állományok másolatait is elraktározza a Rendszer-helyreállító Mappában. Ennek a folyamatnak a során azonban biztonsági ellenőrzés nem történik. Pedig a mai számítógépes kártevők jelentős része féregvírus, vagyis a tcp/ip hálózatokon *(Interneten)* történő terjedésért felelős kód önállóan is működő állomány, amely hagyományos fájlfertőző és/vagy romboló vírustársat hordoz. Érthető okokból a fertőzés kezdetén a Windows gyökér- vagy rendszerkönyvtárába írják ki magukat a kártevők. Egy vagy több új fájl keletkezik tehát, gyakran szándékosan megtévesztő, hasznosságot sugalló névvel *(pl. taskbar.exe)*. A System Restore szolgáltatás e fájlokat rendszerállományoknak tekinti és lementi.

Mentesítsünk!

Az antivírus szoftver ugyan törli a Windows könyvtár alatt felelt kártevőket, viszont a Helyreállító Mappát nem képes elérni, mert ahhoz a LocalSystem jogai sem elegendőek! A SystemFileChecker / DLL-cache szervizek rövidesen érzékelik, hogy a rendszermappából a vírus *(amely számukra csak egy fájl)* törlésre került. Mivel a System Restore együttműködik a fenti szolgáltatásokkal, a helyreállító könyvtárban őrzött víruspéldány automatikusan visszatöltésre kerül. Ezzel versenyhelyzet alakul ki a Windows és az antivírus program között, amelyből a vírus kerül ki győztesként!

A teljes vírus-mentesítéshez *(emberi közreműködéssel)* három út vezethet:

- Más gépbe slave-ként áthelyezzük a merevlemezt és ott mentesítünk. Ezt a megoldást nem tanácsoljuk, mert véletlenül tovább is terjeszthetjük a vírust. Egyes kártevők pedig csak a saját gépben orvosolhatók adatvesztés nélkül *(ilyen pl. a „Magistr.B")*
- Natív DOS módban történő újraindítás, mivel ott jogosultságokról nincs szó, bármi törölhető. XP operációs rendszer esetén azonban az NTFS fájlrendszer nem látható DOS alatt. A Windows ME-ben a DOS mód gyárilag le van tiltva, ez esetben pl. Win98 CD-ről bootolhatunk. Mivel a CAB állományok DOS alatt nem kezelhetők, a vírusmentesítés érdekében kénytelenek vagyunk az egész rendszer-helyreállító mappa archívumot törölni. Ezzel elveszítjük a System Restore által készített érvényes „rendszer-fotókat" is.
- Végül kikapcsolhatjuk a Rendszer-visszaállítás funkciót. A feladat grafikus felületen is elvégezhető, bár a két Windows-ban más-más helyen találjuk ezt a beállítást és hatása csak rendszer-újraindítás után érvényesül. XP esetén a Start Menü, My Computer alatt a Rendszer-tulajdonságok, Rendszer-helyreállítás fülön belül leljük meg a szükséges kapcsolót *(képet ld. a III/6. cikkben)*



♣ ***Win ME / Sajátgépen jobbkattintás / Tulajdonságok / Teljesítmény / Fájlrendszer / Hibakeresés***

Sajnos az Me/XP rendszereken víruskeresők hatékony alkalmazásához egyelőre szükség van a System Restore tiltására, legalább a mentesítés idejére. A Q263455 számú Microsoft Knowledge Base cikkben [3] is ez a megoldás szerepel.

Megemlítendő még, hogy a System Restore, mint háttérben futó szolgáltatás működése egy kicsit több erőforrást emészt fel annál, amit az eredeti cikkben olvashattunk. Gondoljunk arra, hogy a futtatható állományok gyakran eleve tömörítve vannak *(pl. UPX-szel)*, tekintettel a mai OO fejlesztőeszközök által generált „kövér" kódra. A CAB formátum alkalmazása ez esetben nem jelent nagy megtakarítást.

Végezetül megállapíthatjuk, hogy a System Restore szolgáltatás működése valós biztonsági problémákat is felvet. Ezekre a gondokra a Microsoft megfelelően reagálhatna többek közt egy olyan, jól dokumentált interfész kialakításával, amely lehetővé tenné külső biztonsági szoftverek Sytem Restore-hoz illesztését.

Fehér Tamás MCP2000
feher.tamas@2f.hu
www.2f.hu

A cikkben szereplő URL-ek:

[1] http://www.f-secure.com/v-descs/klez.shtml
[2] http://www.microsoft.com/windows/ie/downloads/critical/q323759ie/default.asp
[3] http://support.microsoft.com/support/kb/articles/Q263/4/55.asp

Microsoft Exchange 2000

Exchange Information Store

Az Exchange adattárolás hátterében az Information Store szolgáltatás áll. Ez az egy szolgáltatás - a store.exe - felel az adatok tárolásáért, integritásáért, az adatbázisok épségéért. Mielőtt fejest ugranánk a mélyvízbe, áttekintjük az Exchange adattárolással kapcsolatos tudnivalóit és az újdonságokat. Ezután térek rá az egyes területek részletes ismertetésére, a gyakorlatias oldalra.

Az Information Store

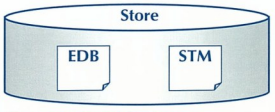
Az Information Store - ahogy az előző verziókban is -lehetővé teszi a tárolt információk MAPI-n keresztüli elérését *(hagyományosan Outlookból)*. Internet Information Serveren keresztül számos más protokoll segítségével is hozzá lehet férni az adatokhoz. Ilyen a HTTP és a WebDAV, vagy az NNTP, a POP3, IMAP4. Ezeken kívül a különböző API-k *(Application Programming Interface-k)* segítségével is hozzáférhetünk az adatokhoz, mint például CDO *(Collaboration Data Objects)*, az ADO *(ActiveX Data Objects)* vagy az ADSI *(Active Directory Services Interface)*. Mindez a sok hozzáférési mód azt jelenti, hogy az Information Store nem egyszerű adattároló, hanem olyan információs központ, amely az adatok széles skáláját képes befogadni, és sokféle módon szolgáltatni.

Web Storage System

A Web Store System egyesíti magában a hagyományos fájlrendszert, a webes és a hagyományos - MAPI-s technikát az adatok tárolására és elérésére. Nem egy speciális vagy új tárolási technológiáról van szó, hanem az adatok szolgáltatásának új koncepciójáról. Csakúgy, mint eddig, az adatok adatbázisokban tárolódnak, és ugyanúgy az Information Store szolgáltatás igazgatja őket. A megközelítés más. Amikor arról elmélkedünk, hogy milyen új webes szolgáltatásokat nyújt az Exchange, akkor Web Store-nak hívjuk a tároló rendszert. De ne keressünk Web Store szolgáltatást a kiszolgálón. A sokféle hozzáférési lehetőség és a sokféle lehetséges adattípus miatt Redmondban az Exchange 2000 Information Store-t elnevezték Web Store-nak. Gyakorlatilag csak Information Store szolgáltatás van, a Web Store csak elméletben létezik. Összefoglaló elnevezés, amely arra utal, hogy az Exchange adatbázisai többre képesek, mint levelek és dokumentumok egyszerű tárolására. Az Exchange 2000 Information Store képes befogadni az összes létező adatformátumot, ráadásul mindezt nemcsak levelező kliensek *(mint pl. Outlook)* számára teszi elérhetővé, hanem böngészők, egyszerű fájlkezelők, vagy más alkalmazások számára is.

Az adatbázisok

Exchange 2000-ben egy adatbázis kettő adatbázisfájlból áll. Az egyik a régről *(Exchange 5.5)* ismert rich-text *(.edb)* fájl, a másik pedig a streaming media *(.stm)* fájl.
Azért van rájuk szükség, mert az elektronikus levelezés teljes egészében az Internet felé fordult, így a ki-be menő levelek zöme MIME formátumú, amelyet az Exchange korábbi verziói beérkezéskor - a tárolás megkönnyítése érdekében - azonnal rich-text formátumúvá alakítottak. Ha azonban egy levél MIME formátumban érkezik, és így kéri a felhasználó is *(pl. Outlook Expressel olvassa)*, a konverzió teljesen felesleges, sőt, dupla! *(Beérkezéskor MIME-> RTF, olvasáskor RTF->MIME)*. De ne szidjuk a Microsoftot: az Exchange legelső *(4.0)* elterjedt verziójának megjelenésekor a vállalati levelezés szabványa egyértelműen az X.400 volt. Internetről nagyon kevesen hallottak még akkoriban!
Az .edb kiterjesztésű fájlban *(rich-text formátum)* vannak azok az állományok - levelek vagy dokumentumok -, amelyek MAPI protokollon keresztül *(tipikusan Outlook használatával)* kerülnek be. Az adatokon kívül az edb kiterjesztésű fájl tartalmazza az adatbázistáblák definícióit, a postaládák, az üzenetek, a mappák szerkezetét.
Az .stm kiterjesztésű adatbázisfájlban az adatok MIME *(Multipurpose Internet Mail Extensions)* formátumban tárolódnak. Ebbe az adatbázisfájlba kerül minden, amit Internetes protokollokon keresztül juttatunk az adatbázisba, pontosabban minden, amit nem MAPI-n keresztül viszünk be.
Ez a két fájl elválaszthatatlan egymástól, mind a kettő szükséges a működéshez, az Exchange 2000 a két adatbázisfájlt egy egységként kezeli. Egy levél vagy az EDB, vagy az STM fájlban csücsül - a születési formátuma határozza meg, hogy melyikben. Ezt a logikai egységet tárolónak *(Store)* hívjuk.



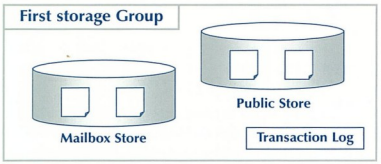
♣ ***A Tároló (Store) logikai felépítése***

Felhasználási szempontból kétféle tárolót különböztetünk meg. A Mailbox Store a postaládák, a Public Store pedig a nyilvános mappák tárolója. Az Exchange 5.5-ben csupán egyetlen postaláda-adatbázis létezhetett egy szerveren, s ez tetszőleges méretűre nőhetett. Igen ám, de egy egybefüggő, 20 gigabájtos tároló szalagos mentése nem kis harci feladat! Exchange 2000 esetén lehetőségünk van több tároló létrehozására ugyanazon a kiszolgálón, ezzel kezelhető méretű tárolófájlokba tudjuk szétosztani a vállalat leveleit. Ezeket együttesen tárolócsoportnak hívjuk *(Storage Group)*.

Tárolócsoportok *(Storage Groups)*

A telepítéskor rögtön létrejön egy First Storage Group nevű tárolócsoport amely magában foglalja a Mailbox Store-t és a

Public Folder Store-t. Kezdetben e két tároló alkotja a tárolócsoportot.

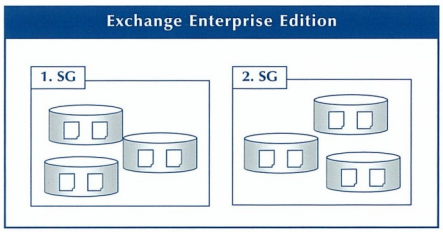


♣ *Alapértelmezett tárolócsoport*

Egy tárolócsoport összes adatbázisa ugyanazt a tranzakciós naplót használja, és egy processzként fut a kiszolgálón. Vagyis egy store.exe egy tárolócsoport összes adatbázisáért felelős. Ennek ellenére az adatbázisokat külön is tudjuk kezelni, menetközben kikapcsolni-bekapcsolni, vagy mentést készíteni róluk.

A tranzakciónapló csoporton belüli közös használatának az az értelme, hogy ez a fájl szekvenciálisan *(a végéhez csatolva)* íródik *(növekszik)*, így ha egyedüli fájlként található egy merevlemezen, az író-olvasó fejet alig-alig kell mozgatni - a naplózás villámgyors. Ha öt tárolót alkotunk, s mindegyiknek saját logja van, öt önálló lemez kellene a naplófájlok magányos növekedésének biztosításához. *(Ha egy lemezre öt naplófájlt teszünk, az író-olvasó fejet bizony annak megfelelően kell rángatni, hogy melyik napló végéhez kell éppen hozzáírni.)* A tárolócsoportok bevezetésével ez a probléma semmissé vált, ugyanis akár öt tároló is osztozhat egyetlen naplófájlon!

Az Exchange 2000 Enterprise változatában lehetőség van arra is, hogy több *(maximum négy)* tárolócsoportot hozzunk létre ugyanazon a kiszolgálón.



♣ *Több tárolócsoport egy kiszolgálón csak Enterprise változat esetén lehetséges*

További korlátozás a Standard változatban, hogy a létrehozott egyetlen tárolócsoportban levő tárolók mérete külön-külön - tehát egy EDB és egy STM fájl együttes mérete! - nem haladhatja meg a 16GByte-ot. Másképpen mondva a Mailbox Store-hoz tartozó priv1.edb és priv1.stm fájlok mérete együttesen maximum 16GB lehet. *(A Public Folder Store-hoz tartozó pub1.edb és pub1.stm fájl együttes mérete is maximum 16GB lehet. A két tároló együtt 32GB-ot érhet el.)*

Az Enterprise változatban az adatbázisok méretét csak a háttértár, illetve az NTFS képességei korlátozzák. A plafon magasan van: 16 exabyte! *(1 exa = 1 giga * 1 giga).*

Több tároló és tárolócsoport létrehozásának lehetősége többféle szempontból is előnyös. A tárolócsoportokat és tárolókat kezelhetjük egyben, vagy külön-külön is. Gondoljunk az Exchange házirendekre, amelyeket tárolókhoz tudunk rendelni. Biztonsági szempontból is fontos lehet különféle adatoknak különböző tárolókat vagy tárolócsoportokat létrehozni. Ha bizonyos adatokon fontos a visszaállíthatóság, másokon viszont nem annyira, megfontolandó, hogy ezeket az adatokat külön tárolócsoportokba tegyük, hogy eltérő tranzakciónaplózást - s ezáltal mentési stratégiát - tudjunk beállítani.

Az egyes tárolókat menet közben is ki-be tudjuk kapcsolni, menteni, törölni vagy újat létrehozni.

Nyilvános mappák

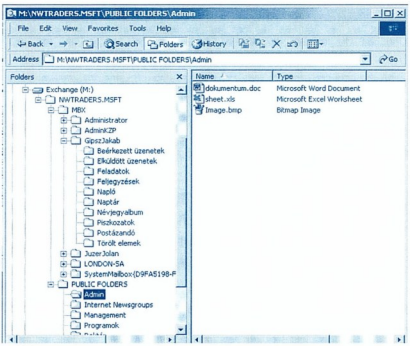
Önmagukban nem hordoznak semmi újat a nyilvános mappák, hisz már az előző Exchange verzióban is léteztek, ott is a közös használatban levő fájlok, közös naptárak tárolására szolgáltak. Ami viszont újdonságot jelent, az a lehetőség, hogy egy kiszolgálón több Nyilvános mappa fa struktúrát is ki lehet alakítani *(Public Folder Tree)*. Az előző verziókban csupán egyetlen ilyen PF struktúra létezhetett, az Exchange 2000-ben viszont minden Public Folder Store-hoz tartozik egy Nyilvános Mappa struktúra. Üröm az örömben, hogy az utólagosan létrehozott PF struktúrák nem érhetők el MAPI kliensekből, tehát például Outlookból sem. Csak a telepítéskor létrejött PF struktúra érhető el az összes protokollon keresztül. *(Tehát hiába van több PF Store, Outlookkal nem tudunk hozzáférni csupán az elsőként, a telepítés során létrejött struktúrához.)*

Felmerül a kérdés miért jó akkor, hogy több tárolót tudunk létrehozni? Azért, mert a második, harmadik stb. nyilvános mappa-tárolóhoz számos más hozzáférési módszer létezik, akár fájlrendszerként, vagy HTTP-n keresztül is használhatjuk.

Installable File System *(IFS)*

Az IFS lehetővé teszi, hogy az Exchange adatbázisban tárolt adatok közvetlenül az operációs rendszerből elérhetőek legyenek *(mint egy csatolt hálózati meghajtó)*. Nem kötelező levelezőklienst használni tehát ahhoz, hogy elérjük a leveleinket, vagy akár a nyilvános mappák elemeit.

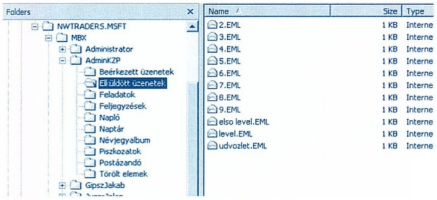
Az IFS telepítés után azonnal használatba vehető: a telepítéskor létrejön a kiszolgálón egy M: betűjelű meghajtó *(ha az M: már foglalt, N: lesz a neve)*.



♣ *Az Exchange adatbázis az Explorerből is elérhető!*

Az M: meghajtót, és mappáit ugyanúgy megoszthatjuk, mint bármely másikat, amit aztán a felhasználóknál egyszerű NET USE paranccsal tudunk csatlakoztatni.
A képen is látható MBX könyvtárban találhatók a postaládák, a PUBLIC FOLDERS alatt pedig a nyilvános mappák tartalma.

A postaládákban található összes üzenetet, feljegyzést, naptárbejegyzést EML kiterjesztésű fájlként láthatjuk az Explorerből.



♣ Elküldött üzenetek az Explorerből nézve

Front-end/Back-end kiszolgálók

Szintén újdonság az Exchange 2000-ben, hogy külön építhetünk olyan kiszolgálókat, amelyek az adatkezelést végzik, és külön olyan kiszolgálókat, amelyek az ügyfelekkel kommunikálnak. Ez azért lehetséges, mert az Exchange 2000-ben szétválasztották az adatkezelést a kommunikációs protokollok kezelésétől.
Az előző verziókban az Exchange maga igazgatta a protokollokat, most viszont mindez az Internet Information Server *(IIS)* dolga. Minden kommunikáció az IIS-en keresztül zajlik. Talán emlékszik az olvasó, a sorozat első cikkében szóba került, hogy az NNTP protokollt az Exchange telepítés előtt fel kell telepíteni. Az NNTP is az IIS része, ahogy az SMTP is. *(Ez utóbbit az Exchange telepítés lecseréli egy robusztusabb verzióra.)*
Az adattárolás és a protokollok kezelésének szétválasztása teszi lehetővé Front-End/Back-End környezet kialakítását. A Front-End kiszolgálókhoz csatlakoznak a felhasználók, a Back-End szervereken pedig az adatbázisok találhatók. *(Az Exchange 2000 Enterprise változat kiváltsága, hogy be lehet állítani Front-End kiszolgálónak.)*
Ilyen esetben a kommunikáció a következő módon zajlik:

1. Az ügyfél a Front-End kiszolgálóhoz csatlakozik egy kéréssel, mondjuk IMAP4 protokollon.
2. A Front-End kiszolgáló eljuttatja a kérést az adatbázist tartalmazó kiszolgálóhoz *(Back-End)*, szintén IMAP4 protokollon
3. Back-End kiszolgáló válaszol a Front-end kiszolgálónak
4. A Front-End kiszolgáló válaszol az ügyfélnek.

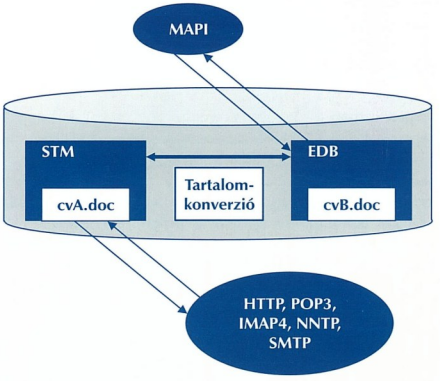
Nagyon jó ötlet ez, hiszen így nemcsak terheléselosztást tudunk biztosítani, hanem a klienseknek sem kell tudniuk, hogy a háttérben tulajdonképp melyik kiszolgálóhoz csatlakoztak. Van viszont egy nagy hátránya: MAPI levelező programokból - tehát az Outlookból - továbbra is közvetlenül a Back-End kiszolgálókhoz kell csatlakozni. A Front-End kiszolgálókhoz csak a többi protokollal *(HTTP, IMAP3, POP3, NNTP)* kommunikáló ügyfél fordulhat.
(Rengeteg dolog nem működik MAPI alól. Folyosói pletykák szerint a MAPI-t halálra ítélték a redmondi fejlesztők, ezért nem készítették fel az Exchange 2000 teljeskörű elérésére. Hogy mi lesz az utódja? Nem tudjuk...)

Full Text Indexing

Bár ezzel a témakörrel még részletesen foglalkozunk egy későbbi cikkben, a teljesség kedvéért itt is megemlítem.
Exchange 2000 előtt nem lehetett teljes szöveges *(Full Text)* indexeket készíteni az Exchange adatbázisokról, a megsokasodott adatmennyiségben sokáig tarthatott egy-egy keresés. A Full Text indexelés alkalmával minden postaládában és nyilvános mappában levő állomány minden szava feltérképezésre kerülhet, legyen akár levél, akár HTML lap, egyszerű szöveg, vagy PDF állomány. Nemcsak a bennük levő szavakra, hanem a fájlok egyéb tulajdonságaira is kereshetünk *(például a szerzőre, a fájl méretére stb.)*.
Redmondban a biztonságra is gondoltak. A keresésekben csak azok az eredmények jelennek meg, amelyekhez egyébként is van hozzáférési jogosultságunk.
Maga az indexelés - különösen nagy adatbázisok esetén - sokáig tarthat, és az indexek meglehetősen sok helyet igényelnek.

Az adatbázisok kezelése

Vegyünk egy nyilvános mappát. Ha Outlookból dokumentumot mentünk bele, az az EDB adatbázisba kerül. Ha OWA-n, vagy az M: meghajtón keresztül tesszük ugyanezt, a fájl az STM adatbázisban landol. Az STM fájlban csak maguk a nyers adatok vannak, a hozzájuk tartozó metaadatok már az EDB-ben. Ebből is látszik: az EDB és az STM fájl nem választható szét, minden esetben ketten alkotnak egy teljes adatbázist.

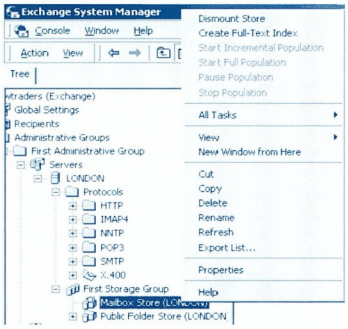


♣ Adatok elérése és konvertálása az elérési protokolloknak megfelelően

Mégis bárhonnan nézzük adatbázisunkat, minden adat azonnal látható. A leveleket éppúgy látjuk az Outlookból, mint a fájlrendszeren keresztül. Ez igaz a nyilvános mappák tartalmára is. Az adatok bárhonnan is kerültek az adatbázisba, mindenhonnan láthatóak, sőt, minden dokumentum csak egy példányban van jelen. Vagy az STM, vagy az EDB adatbázisfájlban.
Ha az M: meghajtón, OWA-n, vagy SMTP-n keresztül érkező fájlt - amelyek az STM fájlba kerülnek - Outlookból szeretnénk olvasni, a hozzáférés pillanatában a kiszolgáló MIME formátumból az Outlooknak emészthető RTF formátumra konvertálja. Mindezt a kiszolgáló a memóriában végzi. Elméletileg. Az összes hozzáférhető dokumentum szerint a konvertált változatot nem írja lemezre, újabb hozzáférés esetén újból konvertál. A gyakorlat azonban mást mutat: nagyméretű fájlok esetén

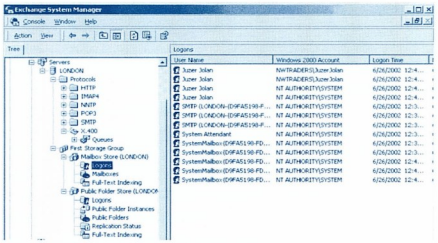
igen könnyen megfigyelhető, hogy valahányszor „idegen" adatbázisban keressük, az adat oda is bekerül, a fájl mérete megnő!
Ugyanez igaz a másik oldalról nézve is. Ha Outlookból nyilvános mappába tett dokumentumot - ami ilyenkor az EDB fájlba kerül - szeretnénk megnyitni közvetlenül az operációs rendszerből, OWA-n keresztül, vagy IMAP4-gyel, a kiszolgáló „röptében" az EDB-ből a megfelelő formátumra konvertálja a dokumentumot a saját memóriájában - majd le is menti sajnos.

A tárolókat külön-külön kezelhetjük, működésük független. Azért nem egészen, mert a store.exe tartja össze az egy tárolócsoportba tartozó adatbázisokat, de ettől még az adatbázisok külön életet élnek.



♣ ***Postaláda tárló kikapcsolása - Dismount Store***

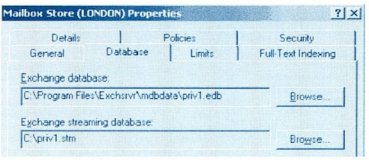
Menet közben a tárolókat egymástól függetlenül ki-be lehet kapcsolgatni. A kiszolgáló csak minket figyelmeztet, hogy dismount alkalmával mindenki leszakad az adatbázisról, a felhasználók csak abból veszik észre a műveletet, hogy nem tudnak hozzáférni többé az adataikhoz. Érdemes tehát óvatosan bánni ezzel a funkcióval, legjobb, ha dismount előtt meggyőződünk arról, hogy senki nem használja az adatbázist. Akár a nyilvános mappák, akár a postaládák tárolójáról legyen szó, a System Managerben az érintett tároló alatt található Logons-ra kattintva láthatjuk a bejelentkezett felhasználókat.



♣ ***Juzer Jolán használja a postaládáját***

Itt nincs lehetőség arra, hogy „kizavarjuk", lecsatlakoztassuk a felhasználót, csak tájékozódhatunk a bejelentkezett felhasználókról.

Az adatbázisok mozgatása
Az adatbázisokat - átmeneti leállás árán - át lehet helyezni más kötetre, ha az adott tároló Database tulajdonságlapján a „Browse..." segítségével kiválasztjuk az új könyvtárat az adatbázisfájloknak, majd az ablak alján az Apply vagy az OK gombra kattintunk. A többit az Exchange elintézi nekünk.



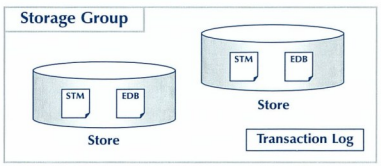
♣ ***Az STM és EDB fájlok mozgatása külön-külön is lehetséges***

Nem szükséges, hogy egy tárolóhoz vagy adatbázishoz tartozó fájlok ugyanabban a könyvtárban legyenek, ez csupán áttekinthetőségi szempontból javasolt. Minden adatbázis minden fájlját oda helyezzük, ahova nekünk tetszik.
Ha a tranzakciónaplót is át szeretnénk helyezni, a tárolócsoporton kísérletezzünk az egér jobb gombjával, hiszen tudjuk: egy tárolócsoport összes adatbázisa közös tranzakciónapló-fájlt használ!

Tárolócsoportok *(Storage Groups)*
Míg az exchange előző verzióiban egy Exchange kiszolgálón összesen két adatbázis létezhetett, - egy adatbázis a postaládáknak, és egy másik a nyilvános mappáknak - addig az Exchange 2000 lehetővé teszi, hogy akár 20 adatbázisunk is legyen egyetlen kiszolgálón. Mindegyik adatbázis egy tárolócsoport vagy Storage group tagja.

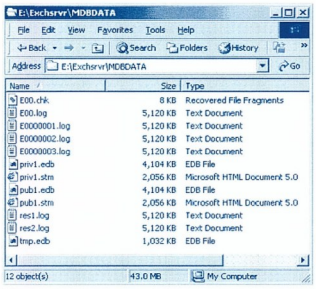
A tárolócsoportok ismertetőjelei:
- A tárolócsoporthoz tartozó adatbázisok ugyanazokat a tranzakciós naplóállományokat használják
- Maximum öt adatbázis lehet egy tárolócsoportban
- Egy adatbázis két fájlból áll - egy edb és egy stm kiterjesztésűből.

Egy tárolócsoport részei láthatók az alábbi ábrán is:



♣ ***A tároló csoport részei***

Az Exchange telepítő létrehoz egy tárolócsoportot - ez a First Storage Group - amelyben rögtön létrehoz két adatbázist is, valamint a tárolócsoporthoz tranzakciónaplókat. Mindezt az \Exchsrvr\MDBDATA könyvtárba helyezi. Ha jobban megnézzük az MDBDATA könyvtár tartalmát, a következőket láthatjuk:



♣ Egy frissen telepített Exchange 2000 „First Storage Group" nevű tárolócsoport könyvtára

Nézzük sorjában a fájlok szerepét:

- Priv1.edb - postaládákat tartalmazó adatbázisfájl - azok az adatok kerülnek ide, amelyeket MAPI protokollon keresztül kezelünk *(tipikusan amikor az Outlookot használjuk)*.
- Priv1. stm - szintén a postaládákat tartalmazó adatbázisfájl, ahol az adatok MIME (Multipurpose Internet Mail Extensions) formátumban tárolódnak. Ebbe az adatbázisfájlba kerül minden, amit Internetes protokollokon keresztül juttatunk az adatbázisba, azaz minden, amit nem MAPI-n keresztül viszünk be.

Ők ketten alkotják a Mailbox Store-t, elválaszthatatlanok.

- Pub1.edb - nyilvános mappákat tartalmazó adatbázisfájl - MAPI-n keresztüli eléréshez
- Pub1.stm - szintén nyilvános mappákat tartalmazó adatbázisfájl

Nem meglepő, hogy ők ketten alkotják a Public Store-t, szintén elválaszthatatlanok.

A tech.net magazin előző számában található részletesebb leírás az adatbázisokról. Érdemes elolvasni!

- E00.log - az épp használatban levő tranzakció napló - minden művelet, amely az adatbázisban történik, először a tranzakciós naplóba kerül, az igazi feldolgozás később, a háttérben történik.
- E00xxxxx.log - a múltban használt tranzakciós napló - a tranzakciós napló fájlok mérete 5MB, ha ez a terület betelik a naplóban, akkor újat kell nyitni, a régi pedig egy hexadecimális sorszámmal ellátva a következő mentésig megmarad - feltéve, hogy nem körkörös naplózást használunk.
- E00.chk - ebben a fájlban tárolódik, mely tranzakciók kerültek az adatbázisba, és melyek várnak feldolgozásra.
- Temp.edb - ideiglenes adatbázis fájl - az épp folyamatban levő tranzakciók tárolására, valamint online adatbázis ellenőrzéskor átmeneti adattárolásra.
- Res1.log, res2.log - helyfenntartó naplófájlok, arra az esetre, ha a partíción elfogyna a szabad hely.

Tranzakció naplókról később részletesen szó lesz.

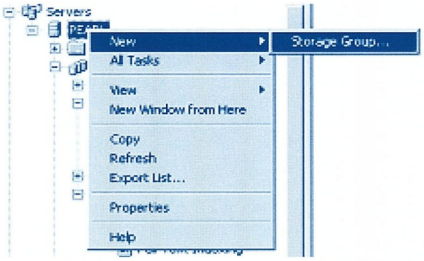
Tárolócsoportok létrehozása

Ahhoz, hogy több tárolócsoportot tudjunk létrehozni, az Exchange 2000 Enterprise változatára van szükség. A Standard változat csupán egyetlen tárolócsoport kezelésére képes, igaz abban akár 5 adatbázist is tárolhatunk.

Nagy különbség az Enterprise és a Standard Exchange között az is, hogy az előbbiben egy tárolócsoporton belül lehet több postaládákat tartalmazó adatbázis is, míg a Standard változat az egyetlen tárolócsoportjában, csupán egyetlen postaládás adatbázist *(Mailbox Store-t)* képes kezelni, a többi négy csak Public Store lehet. Ezt a különbséget láthatjuk az alábbi képen is. A jobboldalon egy Enterprise, a baloldalon egy Standard Exchange 2000 adatbázisai láthatók.

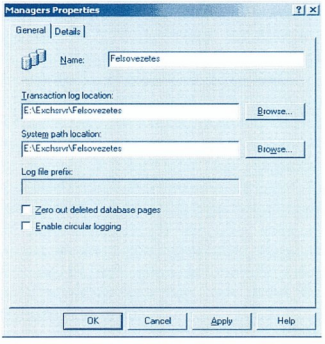
♣ Adatbázisok és tárolócsoportok az eltérő Exchange változatokban

Tárolócsoportot az Exchange System Mangerban hozhatunk létre, a kiszolgáló objektumon állva - Action menü - New - Storage Group-ra kattintva, vagy a context menüből ahogy az alábbi képen is látható:



♣ Egy újabb tárolócsoport létrehozásának első lépése

A következő lépésben meg kell adni a tárolócsoport nevét:



♣ A tárolócsoport elnevezése

Alapértelmezésben a tároló csoportokat a nevüknek megfelelően külön könyvtárakba tehetjük. A cikk elején volt arról szó, hogy egy tárolócsoporthoz egyetlen tranzakciós napló tartozik, akárhány adatbázist is tartalmaz. A tárolócsoport létrehozásakor van lehetőségünk arra, hogy meghatározzuk a tranzakciós napló fájlok helyét. A hibatűrés, és a teljesítmény

szempontjából sem mindegy, hova rakjuk a tranzakciós naplókat, illetve az adatbázisokat.

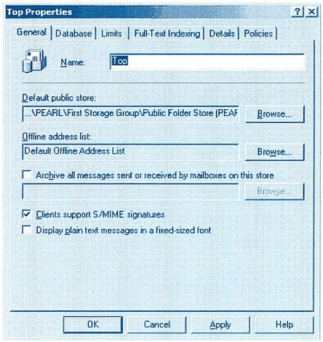
A tranzakciós naplók és adatbázisok elhelyezése

Az Exchange automatikusan - ahogy a képen is látható - az Exchsrvr könyvtáron belül a tárolócsoport nevével megegyező könyvtárba teszi az adatbázisokat és a napló fájlokat is. Ez a lehető legrosszabb megoldás. A tranzakció napló fájlokat lehetőleg elkülönítve tároljuk olyan hibatűrő merevlemezeken, amelyek fizikailag is függetlenek az adatbázis fájlokat tartalmazó szintén hibatűrő merevlemezektől.
Két okból is nagyon fontos, hogy az adatbázist elkülönítve tároljuk a naplófájloktól. Az egyik a hibatűrés, ugyanis, ha külön tároljuk őket, csökkenteni tudjuk a hardver meghibásodás miatti adatvesztést. Például az adatbázist tartalmazó merevlemez(ek) meghibásodása esetén az utolsó mentés és a tranzakciós állományok segítségével helyreállítható az aktuális állapot, ha külön merevlemezeken tároltuk őket. A teljesítmény miatt sem mindegy, hogyan tároljuk az adatbázisokat és a tranzakció naplókat. Minden ami az adatbázisban történik, a tranzakció naplóba kerül és csak később a háttérben futó folyamatként kerül be ténylegesen az adatbázisba. Mindez a gyorsabb kiszolgálás miatt szükséges. Ahhoz hogy a maximális válaszidőt tudjuk megcélozni, a tranzakció naplókat olyan hibatűrő diszk alrendszerre érdemes helyezni, amelynek az írási/olvasási sebessége gyors. A Q319218-as Microsoft tudásbázis cikk **[1]** a következőket írja a tranzakciós naplók és adatbázisok elhelyezéséről:

- Mind az adatbázisokat, mind pedig a tranzakció naplókat tanácsos hibatűrő konfigurációkra helyezni **(RAID5, RAID1, vagy RAID0+1)**
- Ajánlott úgy elkülöníteni az adatbázisokat a tranzakció naplóktól, hogy egyetlen merevlemez meghibásodása egyszerre ne akadályozhassa a naplók és az adatbázisok működését - tehát fizikailag is külön hibatűrő rendszerre érdemes rakni őket
- A postaládás adatbázisokat érdemes külön hibatűrő köteteken tárolni - különösen nagy környezetben.
- Az adatbázist tartalmazó partíciókon a szabad hely mennyisége legyen az adatbázis méretének 110%-a - ez egy esetleges adatbázis visszaállításkor szükséges.
- Csak azután tanácsos újabb tároló csoportot létrehozni, miután ott már nem tudunk több adatbázist létrehozni. *(Emlékezzünk, maximum öt adatbázis lehet egy tároló csoportban.)* Több tárolócsoport esetén több memóriára van szüség a működéshez.

Tehát jól gondoljuk meg, mikor hozunk létre újabb tároló csoportot, hova helyezzük a tranzakció naplókat és az adatbázisokat.
Az adatbázisok létrehozása előtt kell létrehozni a tároló csoportot. Az első adatbázis létrehozásakor jön létre a tranzakció napló is, amelynek helyét nem csak a tárolócsoport létrehozásakor határozhatjuk meg, hanem később is mozgathatjuk. Egy adatbázis mozgatása nem érinti a tárolócsoportjában levő többi adatbázist, de a tranzakcis naplók mozgatásához a tárolócsoportban levő összes adatbázist le kell állítani. Egy postaláda adatbázis létrehozásának lépései a következők:

1. A tárolócsoporton állva a context menü - New - Mailbox Store-ra kell kattintani
2. Az alább ábrán levő ablak fog megjelenni, itt kell megadni az adatbázis nevét, amely automatikusan az adatbázis fájlok neve is lesz:



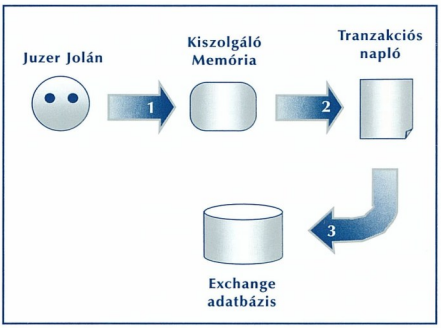
♣ Postaláda adatbázis létrehozása

3. Ha ezután az OK gombra kattintunk, az alapértelmezett beállításokkal létre is hoztunk az adatbázist. Minden egyéb beállítható később is.

Minden adatbázishoz rendelni kell egy alapértelmezett nyilvánosmappa adatbázist *(Default Public Store)*. Elvileg, ha több ilyen nyilvánosmappa adatbázisunk van, akkor az először - az Exchange telepítés során - létrehozott adatbázis helyett választhatunk másik kiszolgálón levőt is.
Az adatbázis Database tulajdonságlapján lehet beállítani az adatbázisok helyét. Legjobb elhelyezni az adatbázist rögtön a létrehozáskor, de természetesen később is változtatható az adatbázisok helye.

Extensibe Storage Engine *(ESE)* az Exchange 2000-ben

Az adatbázisok motorja az Extensible Storage Engine *(ESE)*. Bármilyen változtatás az adatbázisban úgy történik, hogy az ESE a megfelelő lapot az adatbázisból a kiszolgáló memóriájába emeli *(1)*, ott végbemegy a változtatás, amely rögön a tranzakciós naplóba kerül *(2)*, az adatbázis helyett. Az adatbázisok csaknem mindig GB-os méretűek, lassú lenne abban minden művelet esetén megkeresni a lapok helyét. Egyszerűbb, gyorsabb a tranzakciós napló végére beírni az elvégzett műveleteket, később pedig elvégezni az adatbázisban is a változtatásokat *(3)*.



Az ESE dinamikusan foglalja le a műveletekhez szükséges memóriamennyiséget. Pontosabban minden használható memóriát lefoglal, ami a kiszolgálón elérhető, ezzel biztosítja a saját gyorsaságát. Ha egy másik folyamatnak szüksége van

memóriára az ESE csökkenti a saját bufferét dinamikusan. A beépített maximális buffer méret 900Mb egy tároló csoportnak. Mivel minden tárolócsoport adatbázisait külön ESE mozgatja, több tárolócsoport esetén a memóriahasználat is jelentősen megnő, mert mindegyik ESE példány külön foglalja a memóriát.

Az ESE legfőbb feladata a tranzakciók kezelése. Egy tranzakció műveletek sorozatából áll. Egy művelet a legkisebb olyan változtatás, amely egy adatbázison végrehajtható, mint az Insert, Replace, Delete, vagy a Commit.
ESE az úgynevezett ACID tranzakciókkal dolgozik, melyekre a következők jellemzőek:

A, mint Atomic - elemi műveletekből állnak - vagy mindegyik végbemegy, vagy egyik sem.

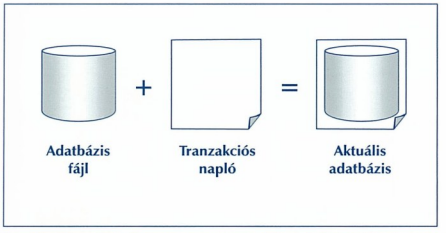
C, mint Consistent - mivel elemi műveleteket hajt végre, ezért az adatbázis egy tranzakció előtt és után is konzisztens állapotba kerül

I, mint Isolated - a változások csak azután láthatóak, miután a tranzakcióban az összes művelet elkészült.

D, mint Durable - az elvégzett tranzakciókat a rendszer tárolja

Az ESE így mindig biztosítani tudja az adatbázis épségét, mert ha egy tranzakciót nem tud teljesen végrehajtani, akkor az egészet visszagörgeti a kiindulópontig. Ebben az ESE segítségére van a tranzakciós napló és a checkpoint fájl.

A tranzakciós naplók rendkívül fontosak az adatbázis helyreállítása, és visszaállítása szempontjából is. Ha mondjuk egy áramszünet miatt a kiszolgáló újraindul miközben rengeteg műveletet végez*(ne)* az adatbázisban, a tranzakció naplóban levő elvégzett műveletek nem vesznek el, hanem a helyreállításnál minden művelet amely még nem érte el az adatbázist, bekerül oda. Az ESE a checkpoint fájlból tudja, melyek azok a műveletek, amelyek már az adatbázisban vannak, és melyek azok, amelyek csak a tranzakciós naplóig jutottak.
A tranzakció napló mérete minden esetben 5 MB. Az épp használatban levő napló neve E00.log. Ha a tranzakció napló betelik, akkor az ESE átnevezi úgy, hogy az E00 mögé egy hexadecimális számot ír, növekvő sorrendben.



Az aktuális adatbázis nem maga a két adatbázisfájl, hanem ehhez az adott pillanatban hozzátartozik az épp használatban levő tranzakció napló *(E00.log)* és valamennyi, az utolsó mentés óta fenntartott naplófájl *(E00xxxxx.log)* is!

Körkörös naplózás *(Circular logging)*

A naplók megmaradnak a következő mentésig, hacsak nem körkörös naplózást használunk.
A naplózás úgy is beállítható, hogy a naplófájlokat újrahasznosítja az ESE. Ilyenkor négy naplófájl van csupán használatban, a legrégebbit mindig felülírja az ESE a következő tranzakciókkal. A checkpoint fájl tartalmazza, mely tranzakciókat kell még az adatbázisban is végrehajtani. Ha túl gyorsan telne a tranzakciós napló, akkor a körkörös naplózásba beszáll egy ötödik naplófájl is. A naplófájlt addig nem tudja az ESE újrahasznosítani, amíg a checkpoint fájlban oda mutató, végrehajtandó tranzakció van.
Alapértelmezésben a körkörös naplózás ki van kapcsolva.
Miért is jó tulajdonképp a körkörös naplózás? Mert helytakarékos. Ezen kívül csak hátránya van. Körkörös naplózás használatakor a legfrissebb adatbázis állapot, amit vissza tudunk állítani, az előző mentéskori állapot. Míg a „rendes" naplózás esetén az utolsó mentésből és az azóta keletkezett tranzakciós naplókból sokkal frissebb adatbázis állapotot tudunk helyreállítani.
Az alábbi táblázatban találhatók, a kétféle naplózási forma tulajdonságai:

Nem körkörös naplózás	Körkörös naplózás
Régi naplófájlok elérhetőek	A naplófájlok újrahasznosulnak
Lehet „differential" mentést is készíteni az adatbázisról	Csak „normal" mentés végezhető az adatbázison
A naplófájlokból és az utolsó mentésből helyreállítható az adatbázis	A naplófájlokat nem lehet a visszaállításhoz felhasználni

Mivel a tranzakcionaplózás nem adatbázisonként, hanem tárolócsoportonként történik, ezért bekapcsolni is tárolós csoport szintjén lehet. Vigyázat! A tároló csoportban levő összes adatbázist érinti a változtatás!

Helyfoglaló naplófájlok

Az ESE tranzakció naplófájlok közt találhatunk két olyat - ez a res1.log és a res2.log - amelyekben nincs semmi, csupán a helyet foglalják egészen addig, amíg a naplófájlokat tartalmazó partíción van elég hely. Normális esetben egy naplófájl betelése után keletkezik a következő. Ha azonban elfogy a szabad terület a partíción, nem marad 5 MB sem egy újabb tranzakciós napló létrehozásához, akkor használja fel az Exchange ezt a két speciális naplófájlt.
A következő történik, amikor nincs mód újabb tranzakciós napló nyitására:

1. ESE figyelmeztetést küld az adatbázis szerviznek
2. Minden memóriában levő tranzakciót megpróbál kiírni a két fenntartott naplófájlba. Sorrendben a res2.logot tölti meg először, majd a res1.logot.
3. Bejegyzi az Event Logba, hogy elfogyott a hely
4. Leállnak az exchange szerizek a kiszolgálón.

Előfordulhat, hogy a 10MB nem elég az éppen futó tranzakciók összes információját lejegyezni, tehát előfordulhat adatvesztés. Érdemes tehát ügyelni arra, hogy mindig legyen elég hely a tranzakció naplóknak és az adatbázisoknak.

Összefoglaló

A több adatbázis és tárolócsoport létrehozásának lehetősége nagy rugalmasságot biztosít számunkra. A tároló csoportokat és adatbázisokat kezelhetjük egyben, vagy külön-külön is. Gondoljunk az Exchange házirendekre, amelyeket adatbázisokhoz tudunk rendelni egyesével, vagy egyszerre többhöz is.
Biztonsági szempontból is fontos lehet különféle adatoknak különböző tárolókat vagy tároló csoportokat létrehozni. Ha bizonyos adatok esetén fontos a visszaállíthatóság másokon viszont nem annyira, megfontolandó, hogy ezeket az adatokat

különböző tárolócsoportokba tegyük, hogy eltérő tranzakció naplózást tudjunk beállítani.
Az egyes adatbázisokat külön tudjuk menet közben is ki-be kapcsolni, menteni, törölni vagy újat létrehozni.
A mentési szempontokat is figyelembe kell venni, amikor tárolócsoportokat hozunk létre. Egy tárolócsoportban egyszerre egy adatbázis mentése lehetséges, ugyanakkor lehet párhuzamosan menteni több tároló csoportot is.
Üröm az örömben, hogy a MAPI-s kliensek *(mikor az Outlookot használjuk)* csupán az először létrehozott Nyilvános mappa adatbázist képesek használni. Az utólag létrehozott nyilvános mappákat tároló adatbázisokat például HTTP, NNTP, vagy egyéb Internetes protokollon keresztül tudjuk elérni - de MAPI-val nem.

Evezzünk most egy kicsit más vizekre, nézzük meg milyen lehetőségek nyílnak az adatbázisokban való keresésre Exchange 2000 esetén.

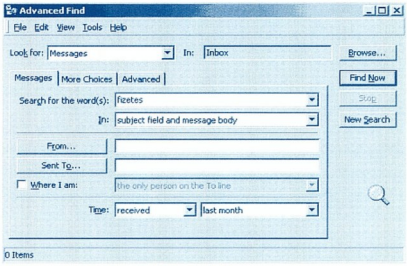
Full Text Indexing

Alapértelmezésben az Exchange 2000 adatbázisban az objektumok tulajdonságai alapján tudunk keresni. Nem újdonság ez, hisz már az előző verziókban is lehetett tulajdonságok alapján kereséseket indítani az adatbázisban. Mit is jelent a tulajdonságalapú keresés? Hétköznapi nyelven azt, hogy az adatbázisban tudunk úgy keresni, hogy egy magadott mezőben keresünk *(mondjuk a Tárgy mezőben)* egy adott kifejezésre. De kereshetünk a feladó, vagy a levél törzsében levő kulcsszó alapján is. Minden kereséskor a postaládában, vagy nyilvános mappákban levő összes üzenetbe bekukkant a keresés a megadott kulcsszót keresve. Attól függően, hogy hány üzenet közt keresünk, ez bizony hosszú-hosszú perceket is igénybe vehet. A tulajdonság alapú keresés ráadásul nem is tud belenézni a fájlokba vagy csatolt dokumentumokba.
Az Exchange 2000-ben használható Full-text indexelés lehetővé teszi, hogy bármely objektum bármely szavára keresve megtalálhatjuk a keresett dokumentumot. Full-text indexing segítségével lehetőség nyílik a csatolt állományokban és egyéb dokumentumokban is keresni.
Ahhoz azonban, hogy ez valóban elérhető legyen, először egy katalógust *(indexet)* kell készíteni. Ez az index tartalmazza a szavakat és a szavakat tartalmazó objektumok helyét az adatbázisban. A kulcsszavas keresés nem közvetlenül az objektumokban történik, hanem a katalógusban, így az eredményt hamar megkapjuk.
A gyors keresésnek azonban ára van. A katalógus építése a processzorokat meglehetősen leterheli. Ezen kívül maga az index mérete nagyjából az adatbázis méretének egyötöde. Ha például egy 10GB-os adatbázist próbálunk indexelni, számíthatunk arra, hogy az index mérete körülbelül 2GB lesz.
Van egy nagy veszélye is ennek a keresésnek. Ha a katalógus nem teljes, vagy helytelen információkat tartalmaz, akkor bizony a keresés eredménye sem lesz megfelelő! Ha full-text indexelést használunk, lehetőség van a csatolt állományokban is keresni. Az index azonban csak Word, Excel, HTML, egyszerű szöveg *(txt)* és beágyazott MIME üzenetek *(.eml kiterjesztés)* szavait tartalmazza.

Full-text indexing és a keresés működése

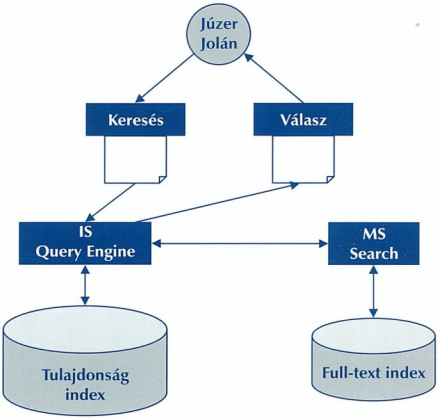
Az indexelést nem maga az Information Store végzi, hanem a Microsoft Search szolgáltatás, ami az Exchange-dzsel egyidőben kerül a kiszolgálókra. A MSSEARCH végzi a katalógusok felépítését és közzétételét az ügyfelek számára. Ezen kívül a katalógusban a konkrét keresést is a Microsoft Search végzi.
Alapértelmezésben a küldő és a címzett, a tárgy mező és a levél törzsében levő adatokból épül fel a katalógus. Egyéb tulajdonságok alapján történő keresést az Information Store maga végzi.
Nézzük a keresést lépésről lépésre.

1. Júzer Jolán elindít egy keresést, mondjuk az utóbbi egy hónapban érkezett levelekre, amelyek tartalmazzák a fizetés szót. Mindezt teszi az Outlook Tools - Advanced Find menüjében. (Figyelem! A full-text keresést csak akkor használjuk, ha kulcsszavakra keresünk!)



♣ *Júzer Jolán keres a levelei közt*

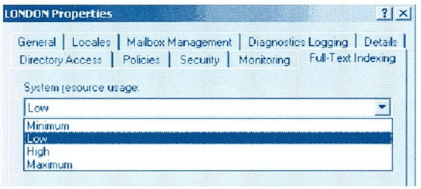
2. Az IS Query Engine-hez jut a kérés, a fizetés kulcsszót továbbítja az MS Search szervíznek.
3. Az MS Search megkeresi a katalógusban a fizetés szót, az eredményt pedig visszajuttatja az IS-hez.
4. Mivel Júzer Jolán csak az utóbbi egy hónap leveleiben keres, ezért a Query Engine a kapott eredményből kikeresi azokat, amelyek az utolsó egy hónapban érkeztek.
5. Végül a Qurey Engine ellenőrzi, hogy Júzer Jolánnak van-e joga a válaszul megmaradt üzenetekre, majd megjeleníti a választ a felhasználónál.



♣ *Katalógusok létrehozása*

Katalógusok nem keletkeznek automatikusan, bár az MS

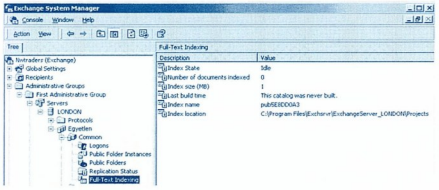
Search szolgáltatás automatikusan elindul. Az indexek létrehozása a rendszergazdák feladata.
A katalógusok egy-egy adatbázishoz kötöttek, szabadon választható, hogy melyikhez szeretnénk a katalógust előállítani. A katalógusok létrehozása a System Managerből lehetséges.
Nulladik lépésként érdemes beállítani, hogy az indexelés mennyire intenzíven foglalhatja le a kiszolgálót. Ezt a System Manager - kiszolgálóobjektum tulajdonságlapjai közt tehetjük meg. Minél kisebbre állítjuk itt az erőforrás-használatot, annál könnyebben elvehetik más folyamatok az indexeléstől a CPU erőforrásait.



♣ Indexelés erőforrás-használatának szabályozása

Abban az esetben, ha nincs más folyamat, amely jelentős processzor munkát igényelne az indexelés annyit használ amennyi neki jólesik, tehát indexelés alatt akkor is lehet a processzor-kihasználtság magas, ha minimumra állítjuk az erőforrás-használatot!
Az index létrehozás lépései:

1. A kiválasztott adatbázis objektumon állva jobb klatty és a menüben megtaláljuk a „Create Full Text Index" parancsot.
2. Megjelenik egy ablakban a katalógus alapértelmezett elérési útja, ami az Exchsrvr \ ExchangeServer_<kiszolgáló> \ Projects könyvtárra mutat. Érdemes megfontolni hová is kerül az index, mert a katalógus mérete nagyjából a teljes adatbázis egyötöde lesz. Hagyhatjuk a beállított értéket, vagy adhatunk más elérési utat is.
3. Az OK megnyomása után létrejön a katalógus alapja a megadott elérési úton. Ennek tulajdonságait meg tudjuk nézni System Managerben:



♣ A frissen létrehozott katalógus tulajdonságai

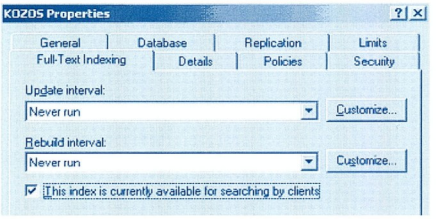
Minden katalógusnak van neve - talán az ábrán is látszik. A megadott elérési útban az index nevének jelzett könyvtárban van tulajdonképpen a katalógus. Minden adatbázishoz különkülön.

A katalógust ugyan létrehoztuk, de nem töltöttük fel. Ez a negyedik lépés.

4. Az adatbázis objektumán állva jobb klatty - Start Full Population parancs elindításával megkezdődik a katalógus felépítése, amely az adatbázis méretétől függően perceket vagy órákat is igénybe vehet.

Amíg a katalógus építése folyamatban van, a fenti ábrán is látható „Index State" állapota „Crawling". Amint befejezte, visszatér „Idle" állapotba.
Miután elkészült a katalógus, elérhetővé kell tenni a felhasználók számára.

5. Az adatbázis Full-text indexing tulajdonságlapján azt az egyetlen jelölőnégyzetet kell bepipálni, ami a képen is látszik. Ezzel elérhetővé tesszük a katalógust a felhasználók számára is.



♣ A katalógus közzététele

Mint ahogy az ábrán is látszik, az adatbázis full-text indexing tulajdonságlapján lehet a katalógusfrissítést is beállítani. Az „Update Interval" adja meg, mennyi időnként frissül az index, mikor kerülnek be újabb adatok. Minél frissebb katalógusban keres a felhasználó annál pontosabb a keresés eredménye, így fontos, hogy jól válasszuk meg azt az időintervallumot, amikor a frissítések történnek.
A „Rebuild Interval" értékét talán hetente egyszeri alkalomra érdemes időzíteni. Ilyenkor a teljes katalógus újraépül, függetlenül az addigi tartalomtól. A teljes újraépítés nagy adatbázisok esetén sokáig tart és sok processzoridőt igényel, ezért nem könnyű olyan időpontot találni, amikor van idő a teljes katalógus felépítésére.

A nyilvános mappákkal folytatjuk…

Dorner Csilla
dorner.csilla@netacademia.net

NetAcademia MesterQrzus Bérlet

Most kivételes akciót hirdetünk. Aki előrelátóan leköti helyét rendezvényeinken, az nemcsak az előadás-sorozatokról nem marad le, de kedvezményesen vehet részt azokon. Az alábbi táblázatba összefoglaltuk akciónk lényegét.

		Akció		
	Teljes ár 1 fő jelentkezése esetén	1. fő	2. fő	3. fő
MesterQrzus 2002. október 31.	**15.000.- Ft**	**15.000.- Ft**	**5.000.- Ft**	**5.000.- Ft**
MesterQrzus 2002. november 28.	**15.000.- Ft**	**10.000.- Ft**	**5.000.- Ft**	**5.000.- Ft**
Mikulás Konferencia 2002. december 6.	**25.000.- Ft**	**15.000.- Ft**	**5.000.- Ft**	**5.000.- Ft**
MesterQrzus 2003. január 30.	**15.000.- Ft**	**5.000.- Ft**	**5.000.- Ft**	**5.000.- Ft**
Összesen(1, 2 illetve 3 fő esetén):	**70.000.- Ft**	**45.000.- Ft**	**65.000.- Ft**	**85.000.- Ft**

Áraink az ÁFÁ-t nem tartalmazzák

Aki a októberi MesterQrzus Kiskonferencián részt vesz, 15.000,-Ft-ot fizet, de ha egyidejűleg jelentkezik az novemberi és januári alkalmakra illetve a Mikulás konferenciára, az összes rendezvényért a 70.000,-Ft helyett csak 45.000,-Ft-ot kell fizetnie.

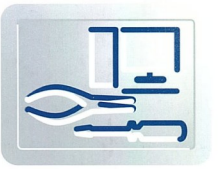
2 vagy több fő jelentkezése esetén az első jelentkező 15.000,-t-ot fizet, minden további jelentkező részvételi díja 5.000,-Ft. Amennyiben cégétől 3 fő jelentkezik a három MesterQrzusra és a Mikulás konferenciára, az első fő 45.000,-Ft-ot, minden további résztvevő fejenként összesen 20.000,-Ft -ot fizet részvételéért.

http://technet.netacademia.net/mesterq

XMLgessünk

XSLT

Az egyik legellentmondásosabb és mégis nagyon sűrűn használt xml technológia az XSLT. Barátkozzunk meg vele!



X aknák

Tisztázzunk néhány X fogalmat. Az XML az Extensible Markup Language rövidítése. Ez definiálja egy XML dokumentum struktúráját **[1]**. Az XML dokumentumok által leírt információk struktúráját, azaz egy XML doksi tényleges információtartalmának szerkezetét az XML Infoset szabvány írja le **[2]**. Ebben a kacsacsőröktől, idézőjelektől és minden egyéb szintaktikai sallangtól mentesen definiálnak egy objektummodellt, amely az xml dokumentumok logikai adatstruktúráját írják le. A jövőbeli szabványok már valószínűleg erre fognak támaszkodni, és végre elszakadhatunk a konkrét szöveges formátumtól.

Az XSL néven jelzett szabvány *(Extensible Stylesheet Language)* valójában három további részből áll **[3]**. Az elsőt nevezik XSLT-nek *(XSL Transformations)* **[4]**, ez xml dokumentumok transzformációjára kidolgozott nyelv. Ez lesz a fő tárgyalási irányunk. Az XSLT-ben szükség van a forrásdokumentum különböző részeinek kijelölésére, erre dolgozták ki az XPath **[5]** szabványt. Amikor XSL transzformációkkal foglalkozunk, leginkább az XSLT és az XPath szabványokat használjuk. Az XSL szabvány harmadik részével, a Formatting Objects-nek nevezett általánosított formátumleírással nem foglalkozunk, mivel a gyakorlatban se nagyon terjed el.

Mire használható az XSL(T)?

Az első időkben az XSL kidolgozásának célja kifejezetten a megjelenítés támogatása, azaz xml dokumentumok valamilyen médiumon keresztüli megjelenítése, formázása volt. A mai webes munkákban továbbra is használjuk ezt az aspektusát, habár *(általában)* a Formatting Objects kihagyásával közvetlenül HTML kimenetet állítunk elő.

Vannak xml alapú grafikus szabványok is, így semmi akadálya, hogy például egy adatbázis kimeneteként előállt xml dokumentumból XSLT segítségével röptében képet generáljuk.

Az XSLT azonban hamarosan önálló szabvánnyá nőtte ki magát, és elkezdték használni különböző xml sémák közötti átjárásra. Tehát nem adatok megjelenítése, hanem az adatstruktúra átalakítása lett a legfőbb csapásirány.

XSLT alapok

Az XSLT egy szabályalapú, deklaratív programozási nyelv. Szabályalapú, mert template-ekkel *(sablonokkal)* leírhatjuk, hogy a forrásdokumentum melyik részét mivé szeretnénk transzformálni. Csak leírjuk, deklaráljuk, hogy miből mi lesz, a transzformáció tényleges folyamát a forrásadatok és az XSLT transzformációt végző motor szabja meg. A sablonok végrehajtási sorrendje nem determinálható az XSLT írásakor, csak futásidőben derül ki. Emiatt ha a sablonok között volnának egymásra hatások, a transzformáció kimenete sokszor nem az lenne, amire számítunk. Ezért az XSLT-t szándékosan mellékhatás-mentesre írták meg, ami olyan, sokszor bosszantó korlátozásokban jelenik meg, mint hogy például a globális változóknak csak egyszer lehet értéket adni.

A deklaratív jelleg igencsak csípi a szemét annak, aki világ életében a C, VB vagy egyéb procedurális nyelveken nőtt fel. Első ránézésre furcsa lesz, de minden korlátozása ellenére az XSLT életképes technológia, amibe úgy tűnik érdemes befektetni.

Helló világ

Első példánkban az alábbi dokumentum lesz a kiindulásunk:

```
<?xml version='1.0' encoding="ISO8859-2" ?>
<hello>
  <pajti>Soci</pajti>
  <duma> dv zlet a vilÆgnak!</duma>
</hello>
```

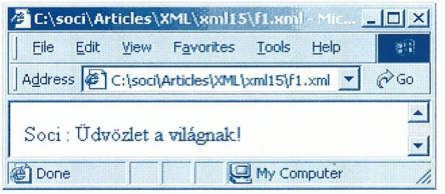
Az alábbi transzformációt fogjuk rá alkalmazni:

```
1 <?xml version="1.0" encoding="ISO8859-2" ?>
2 <xsl:stylesheet version='1.0'
  →xmlns:xsl='http://www.w3.org/1999/
  →XSL/Transform'>
3   <xsl:template match="/">
4     <xsl:value-of select="hello/pajti" />
5     :
6     <xsl:value-of select="hello/duma" />
7   </xsl:template>
8 </xsl:stylesheet>
```

A teszteléshez első körben az Internet Explorert fogjuk használni, amely képes xslt transzformáció közvetlen végrehajtására, ha azt hozzáláncoljuk a forrásdokumentumhoz. Ehhez az xml forrásunkba be kell szúrni egy vezérlőutasítást az xml deklaráció és a gyökérelem közé:

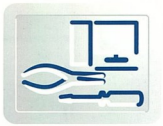
```
<?xml version='1.0' encoding="ISO8859-2" ?>
<?xml-stylesheet type="text/xsl" href="t1.xsl" ?>
<hello>
```

Ezek után már csak be kell töltenünk a forrásdokumentumot az IE-be:



Az Explorer felolvassa a forrásdokumentumot, betölti a hozzá tartozó transzformációt, végrehatja azt a forráson, és mi már a transzformáció kimenetét láthatjuk a böngészőablakban.

Néhány szó a szükséges szoftverekről. A transzformációk helyes működéséhez legyen fenn a gépen az MSXML3 SP2 *(vagy a legutóbbi)* programcsomag is. Van már újabb xml par-



ser csomag is az MSXML4 személyében, azonban az előbbi módon végrehajtott transzformációk ezt nem tudják kihasználni. Csak kézi, kódból végrehajtott transzformációban, explicit 4-es verziójú objektumpéldányok létrehozásával lehet élni az új lehetőségekkel *(pl. XSD)*. Általában érdemes mindkét csomagot felrakni, és kódból az újabb verziót használni - az Explorer meg elvan a 3-assal. De térjünk vissza a transzformációnkra! Az első sor közönséges xml deklaráció, mivel az XSLT nem más, mint egy xml dialektus. A második sor a transzformáció gyökéreleme, ami stylesheet vagy transform lehet, a kettő egyenértékű. Figyeljük meg, hogy az XSLT nyelvtanához tartozó elemek *(a transzformációs nyelvtan)* az xsl prefixel ellátott névtérbe vannak rendezve. A prefix neve tetszőleges, a transzformációt a hosszú URI azonosítja. Fontos, hogy az URI-t betűről-betűre pontosan írjuk le, különben nem fog működni a transzformáció. Az elírás legbiztosabb jele, ha a transzformáció kimenete maga az XSLT tartalma.

Az összes, kimenetet előállító parancsot xsl:template elemek között helyezzük el, ez a transzformáció egysége, blokkja. A match attribútumban kell megadni, hogy a sablon a bemeneti dokumentum mely részét dolgozza fel. Itt egy XPath kifejezést vár az XSLT processzor. A / a teljes dokumentumot jelenti. Egy normál XSLT-ben általában több sablon is helyet kap, mindegyik más match attribútummal. A match="/" sablon kiemelt jelentőségű, mert ő fut le mindig először. Tekinthető a feldolgozás belépési pontjának, ő a main metódus.

A forrásdokumentumból az xsl:value-of elem segítségével írathatunk ki részeket a kimenetbe. A select attribútumban kell megadni a kiíratandó XPath kifejezést. A hello/pajti egy relatív XPath kifejezés, amely abszolút eléréssel így néz ki: /hello/pajti. A sablonok esetén értelmezhető egy XPath útvonal, amit éppen feldolgoz az adott sablon. Ezt hívják Current Contextnek. Esetünkben ez a /, így adódik ki a relatív utunk teljes alakja.

Az ötödik sorban szereplő kettőspont neve string literal. Nyugodtan vegyíthetünk szövegeket vagy akár xml elemeket is a generált kimenetbe, ezek egyszerűen beleszövődnek a dinamikusan generált tartalomba.

Tegyük fel, hogy a nevet kissé bonyolultabban akarjuk megformázni, például vastag betűvel akarjuk szedni. Ehhez html elemek közé kell rakni a value-of kimenetét. Ezt megtehetjük a gyökérsablonban is, de a könnyebb olvashatóság miatt tegyük ezt ki egy újabb sablonba:

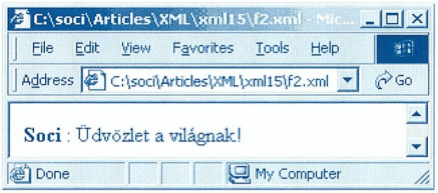
```
<xsl:template match="pajti">
  <b><xsl:value-of select="." /></b>
</xsl:template>
```

Látható, hogy ez a sablon a „pajti" nevű elemeket dolgozza fel. Kiíratjuk a Current Context által meghatározott elem tartalmát, amire XPath-ban a „."-tal lehet hivatkozni *(hasonlóan a fájlrendszer-könyvárakhoz)*.

A példát kipróbálva azonban semmi változást nem fogunk tapasztalni. Azért nem, mert az új sablonunk nem fog „csak úgy" elindulni. Tehát az XSLT nem úgy dolgozza fel a forrást, hogy végigfut a forrásdoksin, és minden egyes node-ra megnézi, hogy van-e egyező sablon. Nem, nem így működik. Megnézi, hogy van-e gyökérsablon, ha van, azt meghívja. Ha a gyökérsablon át akarja adni a vezérlést további sablonoknak, azt explicit meg kell tennie! Erre szolgál az xsl:apply-templates elem:

```
<xsl:template match="/">
  <xsl:apply-templates select="hello/pajti" />
```

Az apply-templates select attribútumában újfent egy XPath kifejezést kell megadnunk. Ez kiválasztja a hello elem pajti nevű gyermekelemét. Egy olyan node halmaz jön létre a memóriában, amiben ez az egyetlen pajti elem lesz. Ezek után az apply-templates úgymond felajánlja: „ki az a sablon, aki tud valamit kezdeni ezzel a node halmazzal, ami egy pajti elemből áll?" Az új sablonunk boldogan jelentkezik, így a vezérlés átkerül hozzá. Ebben a sablonban a Corrent Context már az egyetlen pajti elemünk lesz, ezért nem kell már a value-of-nak tovább navigálni. Látható, hogy ebben az esetben nem html formázó elemet is generálunk a kimenetbe, aminek a hatása azonnal érzékelhető lesz:



Mi történik, ha a forrásban több, mint egy pajti elem van?

```
<hello>
   <pajti>Soci</pajti>
   <pajti>Betti</pajti>
   <pajti>Fifi</pajti>
   <duma> dv zlet a vilÆgnak!</duma>
</hello>
```

Ha az előbbi működési módra visszagondolunk, az apply-templates most egy három pajtielemből álló nodehalmazt válogat le, ezért a pajti sablonunk háromszor fog meghívódni:

```
SociBettiFifi :  dv zlet a vilÆgnak!
```

Mindenféle ciklusszervezés nélkül könnyedén végigmehetek tetszőleges számú node-on, és formázottan jeleníthetem meg őket. Ez eléggé furcsa a proceduráis technikához szokott fejeknek, hisz ott azzal kezdenénk, hogy írjunk egy ciklust, ami bejárja az elemek halmazát. Ha valakinek ez nagyon hiányzik, megvan rá a lehetőségünk:

```
<xsl:template match="/">
  <xsl:for-each select="hello/pajti">
    <b><xsl:value-of select="." /></b>
  </xsl:for-each>
```

Most a for-each elem belsejében található tartalom hajtódik végre a select attribútumban kijelölt node-okra. Kimenete azonos az előbbi példáéval.

Mind az apply-templates, mind a for-each dokumentum sorrendben generálja le a node listát, így a kimenet ennek megfelelő lesz. Gyakori, hogy ez nekünk nem jó, például névsorban szeretnénk látni a pajtikat. Mi sem egyszerűbb:

```
    <xsl:for-each select="hello/pajti">
      <xsl:sort select="."/>
      <b><xsl:value-of select="." /></b>
    </xsl:for-each>
Kimenet:
BettiFifiSoci :  dv zlet a vilÆgnak!
```

Beraktunk egy sort nevű XSLT elemet a ciklus elejére, és a select attribútumában megadjuk azt az elemet, amire rendezni kell a bejárandó node halmazt. Esetünkben ez a pajti elem tartalma, amire a „." hivatkozik *(a template mellett a for-each is változtatja a Current Contextet, így a „." mindig az aktuálisan bejárt elemet adja vissza).*
Az előbbi rendezésben a rendezendő adat típusa string volt. Ha például az alábbi példát

```
...
<hello>
   <pajti kor="67">Soci</pajti>
   <pajti kor="21">Betti</pajti>
   <pajti kor="4">Fifi</pajti>
</hello>
```

a kor attribútumot, mint számot értelmezve akarjuk rendeztetni, a következőképpen kell a sortot paraméterezni:

```
    <xsl:for-each select="hello/pajti">
      <xsl:sort select="@kor" data-type="number"/>
      <b><xsl:value-of select="." /></b>,
    </xsl:for-each>
Kimenet:
Fifi, Betti, Soci
```

A @kor a kor nevű attribútumra utal, a data-type pedig arra utasítja az XSLT motort, hogy próbálja meg számmá alakítani a hivatkozott kifejezést, és annak megfelelően rakja sorba az értékeket. Sajnos a sort csak a szöveges és a numerikus típusokat ismeri, például dátumot nem. Ennek egyszerű oka van. Az XSLT és az XPath kidolgozásakor még nem volt általánosan elfogadott XML típusrendszer, amit most az XML Schema *(XSD)* szabvány testesít meg. Viszont minimális típusokra szükség volt, ezért az XPath szabványban az alábbi típusokat definiálták: szöveg, szám, bool, nodehalmaz, xml-dokumentumtöredék. Semmit dátum vagy egyéb hétköznapi típus! Valószínűleg az XPath 2.0 szabvány már az XSD típusaira épít, és innentől kezdve (XSLT 2.0) a sort is használható lesz bármely sématípusra. Addig együtt kell élni ezzel *(és még számtalan más)* korlátozással.
Néhány gondolat a további sort-paraméterekről. Az order={"ascending" vagy "descending"} attribútummal állíthatjuk be a kimeneti sorrendet. Nekünk magyaroknak érdekes lehet még a sort lang attribútuma. Az alábbi töredéket rendeztetve

```
  <edesseg>Csoki</edesseg>
  <edesseg>Cukor</edesseg>
```

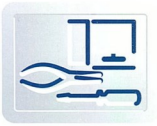
ezt kapjuk *(legalábbis angol system default locale mellett)*:

```
Csoki, Cukor,
```

ami nyilvánvalóan nem helyes a magyar helyesírás szabályai szerint, hisz a „cs" a „c" után jönne. Ebben segít a lang attribútum:

```
<xsl:sort select="." lang="hu"/>
```

Így a rendezés már magyar logikával fog működni.



Egy kis tesztelési útmutató

Az eddigi példákat böngészőben próbáltuk ki, de bonyolultabb XSLT-knél nem mindig kapunk egyértelmű jelzéseket hibák esetén, ezért érdemes megtanulni az MSXSL használatát **[7]**. Ez egy konzolalapú xslt-tesztelő alkalmazás. Használata roppant egyszerű:

```
Msxsl.exe forrÆs.xml transzformÆci .xsl
```

(Az msxsl nem értelmezi a forrásdoksi <?xml-stylesheet ...> vezérlő utasítását.)
Az előbbi példánk kimenete így néz ki msxsl-lel:

Mik azok a furcsa üres helyek a karakterek között? A válasz az első sor encoding attribútumában keresendő. Alapértelmezett módon a transzformáció kimenete UTF-16 kódolású, azaz egyfajta unicode formátumú. Ez sokszor nem kényelmes, valamilyen karakterenként egy bájtot használó kódolásra lenne szükségünk. Ezt az output xslt elemben adhatjuk meg:

```
<xsl:transform version='1.0'... >
  <xsl:output encoding="ISO-8859-2"/>
```

Így a kimenet már a megadott kódolású lesz:

Természetesen a kimenetet gyakran átirányítjuk fájlba, így könnyebb kielemezni, azt kaptuk-e, amit terveztünk.

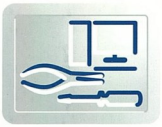
Fejlettebb XSLT

Az alapokat már ismerjük, itt az ideje néhány fejlettebb technikát is tanulmányoznunk.
Láttuk, hogy mind for-each, mind template-ek segítségével fel tudunk dolgozni node setet, így adódik a kérdés, melyiket használjuk? Aki csak proceduralisan hajlandó gondolkodni, az természetesen a for-each-et fogja választani. Azonban *(általában)* a for-each sokkal érzékenyebb a transzformálandó dokumentum szerkezeti változásaira, mind a template-es megoldások. Például az

```
  <xsl:for-each select="hello/edesseg">
```

csakis a hello elemek alatti edesseg elemeket hajlandó feldolgozni, más szinten levőket nem. Ezzel szemben template-ekkel sokkal rugalmasabb transzformációkat is írhatunk, amelyek nem annyira érzékenyek a bemeneti struktúrára.
Egyből ugorjuk bele a lehető legflexibilisebb template-példába:

```
<xsl:template match="/ | @* | node()">
  <xsl:copy>
    <xsl:apply-templates select="@* | node()"/>
  </xsl:copy>
</xsl:template>
```



Vadul néz ki, ugye? Engedjük rá erre:

```
<?xml version='1.0' encoding="ISO8859-2" ?>
<hello>
   <edesseg>Csoki</edesseg>
   <edesseg>Cukor</edesseg>
</hello>
```

Kimenet:

```
<?xml version="1.0" encoding="ISO-8859-2"?><hello>
   <edesseg>Csoki</edesseg>
   <edesseg>Cukor</edesseg>
</hello>
```

Ejha, ez nagyon hasonlít a bemenetre! Sőt, szinte megegyezik, csak a hello elem átköltözött az első sorba.

Helyhiány miatt nem mutatok be más példákat, de ez a transzformáció bármely bemenetet képes megismételni, ezért is hívják identitás-transzformációnak.

Hogyan működik? Intuitíven megfogalmazva rekurzívan bejárja a forrás összes xml konstrukcióját, és egyesével átmásolja őket. Közelebbről ez úgy néz ki, hogy az első és egyetlen template-ünk képes feldolgozni a dokumentum kezdetét *(/)*, valamint harap az összes attribútumra *(@*)*, és az összes node-ra *(node())*. A node() hatókörébe nem tartoznak bele az attribútumok, ezért kellett őket külön kiírni *(XSL Patternsben még nem így volt, MSXSL 2.6-ban (IE5) még nem így működött!)*. A | *(pipe)* jel a halmazok unióját jelenti.

A dokumentum kezdetén tehát elindul a template. A copy elem arra utasítja a processzort, hogy másolja át a kimenetbe az aktuális node-ot *(a Current Contextet)*. Ráadásul a copy úgy van megalkotva, hogy amellett, hogy átmásolja az aktuális node-ot, még belemásolja a törzsében megadott tartalmat is. Azaz első körben átmásolja azt, hogy elkezdődött a dokumentum *(kimegy az xml deklaráció)*.

De mi lesz a belsejében? Az apply-templates kiválasztja az adott szinten *(azaz dokumentumszinten)* az összes node-ot vagy attribútumot. Ebbe beleférnek a vezérlőutasítások, kommentek és maga a gyökérelem is. Attribútum ezen a szinten nincs, hisz az attribútumok elemekhez tartoznak, és most még nem fúrtunk le egy elemig.

Tehát az előbbi példánkra alkalmazva az apply-templates által összegyűjtött node halmazban a hello elem lesz. Ki fogja feldolgozni az apply-templates által felajánlott node-ot? Nos, egy olyan template, aki képes feldolgozni a hello nevű, elem típusú node-ot. Ki lesz az? Természetesen az egyetlen template-ünk, hisz node() egyezést mutat az elemünkkel. Ezért újra elindul a sablon. A copy átmásolja a hello-t a kimenetbe, így ezen a szinten így nézne ki a kimeneti dokumentum:

```
<?xml version="1.0" encoding="ISO-8859-2"?><hello
```

Szándékosan nem zártam le a hello elemet, mert lehet, hogy még vannak attribútumai. Ha vannak, a copy, a sablon és a @* együttműködésével azok is kikötnek a kimenetben, a hello elem belsejében.

Azt hiszem nem kell további boncolgatnunk a példát, lehet érezni, hogy az indentitás-transzformáció a forrásdoksit csavarjaira bontja, hogy aztán újra, a megfelelő sorrendben összerakhassa. A hello elől a soremelés azért tűnt el, mert az xml parser a forrás beolvasása közben eldobja a nem szignifikáns *(lényegtelen)* whitespace karaktereket, azaz azokat, amelyek nem hordoznak információt. A kacsacsőrös, serializált xml formátumból memóriabeli infosetet épít, amiben már nincsenek benne sem a kacsacsőrök, sem a lényegtelen whitespace-ek. Látszólag az identitás trafó akadémiai példa, azaz semmi haszna. Azonban nagyon jó kiindulási alap olyan transzformációkhoz, amelyekben a forrásadatok nagyrészét változatlanul kell átvinni a kimenetre, csak esetleg egyes faágakat ki kell szűrni, vagy egyes elemeket át kell nevezni.

Ha például egy vásárlókat, és a hozzájuk tartozó megrendeléseket tároló xml struktúrából ki szeretnénk szűrni az 1997-nél régebbi megrendeléseket, ki kell egészíteni az identitás-transzformációt az alábbi sablonnal:

```
<!-- Ezek __NEM__ lesznek benne a kimenetben -->
<xsl:template match="orders[number(substring(
 →OrderDate, 1, 4)) &lt; 1997]">
 <xsl:comment>Teszt komment, ez jelenik meg
 a kiszurt node-ok helyÖn -->
 </xsl:comment>
</xsl:template>
```

Ez a sablon figyel a []-ben leírt feltételnek megfelelő orders elemekre. Amikor egy ilyenhez ér az identitás-transzformáció apply-templates eleme, mindkét sablon felhatalmazottnak érzi magát, hogy lekezelje az orders elemet. Ilyen ütközéseknél a konkrétabb match kifejezést alkalmazó template kapja meg a vezérlést, azaz az új sablonunk. Az meg egyszerűen lenyeli a teljes elemet, azaz nem hívja meg rekurzívan a korábbi másolósablont. Hát nem varázslatos? És ez még csak a kezdet...

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

A cikkben szereplő URL-ek:

[1] Letölthető példakódok
http://technet.netacademia.net/download/xml
[2] XML Infoset
[3] XSL szabványhttp://www.w3.org/TR/xsl
[4] XSLT szabványhttp://www.w3.org/TR/xslt
[5] XPath szabványhttp://www.w3.org/TR/xpath
[6] MSXML4
[7] MSXSL

.NET Akadémia

.NET Remoting

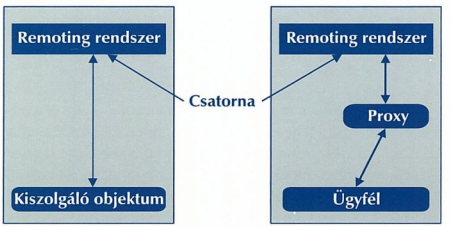
Igen gyakori feladat programok közötti kommunikáció megvalósítása. Nem kell rögtön elosztott alkalmazásokra gondolni, egy egyszerű Szerviz adminisztrációját is érdemes külön programként megvalósítani, és akkor mindjárt szükség van processzek közötti kommunikációra. A remoting az ilyen esetekre nagyon könnyen használható módszert ad a kezünkbe. A témakör komplexitása miatt az a .NET Akadémia rész jelentősen bővebb a szokásosnál, de a kérdéskör hasznossága miatt mindenképpen megéri a fáradságot végigtanulmányozni.

Miről is szól ez a remoting?

A szó az angol remote, azaz távoli szóból ered. A jelző távoli objektumokra utal, azaz a remoting egy olyan technológia amely segítségével nem a mi folyamatunkban futó objektumokat tudunk létrehozni és azok metódusait meghívni.
Láttunk már eddig is ilyet. A DCOM pont ugyanerről szól, csak COM világban megvalósítva. A Java RMI, Remote Method Invocation szintén a .NET remotinghoz hasonló technológia. A CORBA szintúgy, ez is leginkább Java környezetben használatos.
Kódra fordítva valami ilyesmire kell gondolni:

```
TÁvoliObj o = KØrekEgyTÁvoliObjektumot("Az obj
neve", "A tÁvoli gØp adatai");
o.Met dus(paramØter1, parmØter2, ...);
```

A TavoliObj egy közönséges osztály. A KérekEgyTávoliObjektumot bűvös függvény képes arra a csodára, hogy például egy másik gépen létrehozza a kívánatos objektumot, és visszaad nekünk valamit. De mit ad vissza? Nyilván nem a TávoliObj egy példányát, hisz az azon végzett metódushívások a saját gépünkön zajlanának le. Ehelyett visszakapunk egy maskarás objektumot, ami pont úgy néz ki, mint a TávoliObj. Ez azt jelenti, hogy pont ugyanolyan metódusai, jellemzői, stb. vannak neki, mint a TávoliObjnak, de ő nem egy TávoliObj típusú objektum. Ő csak egy átjátszó, aki arra képes, hogy a rajta végzett metódushívásokat átjátssza a másik gépen létrehozott valódi TávoliObj objektumpéldányra. Az ilyen álarcos objektumokat hívják proxyknak. Természetesen a másik oldalon is kell valaki, aki pedig várja a proxy objektum kéréseit, őt hívják a COM világban stubnak *(tönk)*. .NET-nen a tönköt a CLR valósítja meg.



♣ *1. ábra: Remoting architektúra*

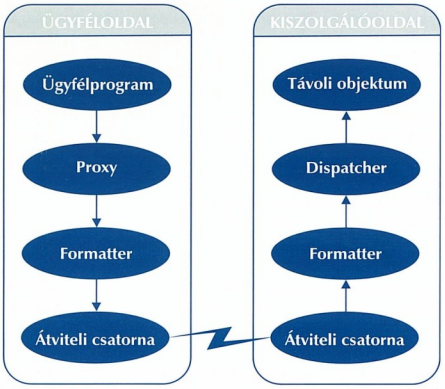
Csatornák

Hogyan tud kommunikálni két program? Alapvetően valamilyen hálózati protokollra lenne szükség ha gépek közötti kommunikációra vágyunk, vagy valamilyen interprocessz kommunikációra, ha egy gépen futó különböző processzek között kell kommunikálni. Valójában menedzselt programokban egy win32 processz is tovább van osztva Application Domainekre, és a Remoting AppDomainek közötti kommunikációra szolgál. Azaz egy win32 processzben lakó két AppDomain is csak Remoting segítségével képes kommunikálni. De számunkra ez a tény most nem túl fontos, mi úgyis processzek és gépek között fogunk csevegni.
Tehát kell valamiféle médium, ami átviszi a hívások tényét és paramétereit, valamint visszaszállítja visszatérési értékeket és a kimeneti vagy referenciális paramétereket. A .NET Remoting Channeleknek, csatornáknak nevezi eme közegeket. Ez nem föltétlen egy egyszerű hálózati protokoll csatornáját jelenti, hanem lehet például egy Message Queue, vagy egy SMTP levél is. Alapban két csatornatípust kapunk a .NET keretrendszerrel: egy TCP és egy HTTP csatornát. Nyilvánvaló, hogy a HTTP TCP-ben utazik, de a kettő közötti különbség, hogy a HTTP esetén a kommunikáció a HTTP szabványnak megfelelően működik, így könnyen összebarátkoztatható tűzfalakkal és proxy szerverekkel *(ne keverjük a proxy szervert a korábbi proxy objektummal)*.
A .NET Remoting architektúra extrém módon bővíthető, gyakorlatilag bármely komponensét ki lehet cserélni. Emiatt fellelhető az interneten olyan Remoting Channel implementáció is, ami Message Queue-t használ átviteli közegnek.

Paraméterek formázása

Milyen formátumban utaznak a hívási paraméterek a kiválasztott csatornán? Gondoljuk azt át, hogy a csatorna fajtája és a paraméterek kódolási formátuma abszolút független egymástól. Ez egyik csak megformázza az adatokat, a másik meg bután elviszi a másik oldalra, hogy aztán ott megint vissza lehessen őket alakítani. A hívott metódusok paramétereiként használt típusokat a Formatterek alakítják át adatfolyammá *(bájt vagy karakterhalmazzá)*.
Alapban kétféle Formattert kapunk: a BinaryFormattert és a SoapFormattert. Az előbbi a lehető legtömörebb módon ábrázolja a típusokat, például egy 32 bites egészet 4 bájton fog ábrázolni, és a kimenete egy bájtfolyam. A SoapFormatter szöveges kimenetet állít elő, és a típusokat a SOAP szabvány encoding szekciójának megfelelően fogja megformázni.
Van tehát kétféle csatornánk, és kétféle paraméter formázónk. Mind a négyféle kombinációban lehet őket használni, ám alapértelmezett módon a TCP csatorna a bináris formázót

használja, a HTTP csatorna pedig a SOAP-ot. Mint tudjuk a SOAP csomag egy szószátyár xml dokumentum, így a HTTP-SOAP kombináció a leglassabb az összes kombináció közül. Viszont a SOAP könnyen értelmezhető formátuma miatt a tűzfalakon könnyű megregulázni, hogy ki, milyen metódusokat hívhat meg SOAP segítségével.
Ha fontos a tűzfalon keresztüli kommunikáció, de nem fontos a szűrés, akkor be lehet vetni a bináris formázót HTTP csatornán is. Ez hasonló hatékonyságú tud lenni, mint a TCP-Bináris kombináció, de mégis átmegy a tűzfalakon. Nagy előnye ennek, hogy elég gyors, és az SSL segítségével még titkosítást is nagyon egyszerű alárakni.
Ha cégen belüli kommunikáció esetén a sebességre kell koncentrálni a más rendszerekkel való együttműködés és a tűzfalak rovására, akkor egyértelműen a TCP-Bináris formázó kombinációt kell használni.
A negyedik, TCP-SOAP kombináció használható, csak nincs túl sok értelme. Lassú, és a tűzfalak is utálják.
Most, hogy ismerünk minden komponenst, lássuk az előbbi ábrát egy kicsit kibővítve.



♣ *2. ábra: Remoting komponensek*

A dispatcher komponens hívja meg ténylegesen a távoli objektum metódusait, tulajdonképpen ő szimulálja az ügyfélprogram hívását. Ezt is automatikusan biztosítja a CLR, nincs vele gondunk.

Paraméterátviteli módok

Az előbbi részben egyféle paraméterátviteli módot mutattam be, amelyet Marshal-by-Value-nak hívnak. A Marshal szó rendezést, elrendezést jelent, és a paraméterek átküldésére utal. A Marshal by Value *(rövidítve: MBV)* azt jelenti, hogy a paraméterek a választott formatter segítségével bájtfolyammá alakulnak, amelyeket a másik oldalon visszaalakítanak élő objektummá *(másolattá)*. Nézzük meg az alábbi példát:

```
TÁvoliObj o = ...;
int[] t = new int[1000000];
o.Met dus(t);
```

A Metódus paraméterül kap egy 4 MBájt méretű tömböt. Az előbbieknek megfelelően ha az int tömb egy MBV objektum *(tényleg az)*, akkor az ügyfél oldali formatter átalakítja egy legalább 4 MBájt méretű bájtfolyammá, átküldi a csatornán *(pl. modem)*, a másik oldalon pedig visszaállítják a tömböt, és elvégzik a metódushívást.
Hogyan történik meg a paraméterek bájtolyammá alakítása? Hogyan működnek a formatterek? A kulcsszó a Szerializálás. Egy objektum *(objektum-gráf)* adatfolyammá alakítását hívjuk Szerializálásnak, a visszaalakítást pedig Deszerializálásnak. A formatterek automatikusan képesek bármely objektum Szerializálására és Deszerializálására, ha az adott típus meg van jelölve a [Serializable] attribútummal:

```
[Serializable]
class Kutya {
  string neve;
  int kora;
}
```

Ennyi az egész. A formatterek Szerializáláskor nem tesznek mást, mint reflection segítségével végigolvassák az adott objektum tagváltozóit, és a saját formai szabályaik alapján átalakítják őket bájtfolyammá. Deszerializálás esetén pedig létrehoznak az objektumból egy példányt, majd a kapott bájtfolyamból olvasgatva kitöltik a mezők értékét. Azaz a Kutya objektumot paraméterként használva a kutya példány nevét és korát a formatter összecsomagolja, amit a másik oldali formatter alakít át egy másolat kutya objektummá:

```
Kutya k = new Kutya();
o.Met dus(k);
```

A szerializálás folyamata további attribútumok segítségével szabályozható, illetve készíthetünk teljesen egyedit is az ISerializable interfész és egy plusz konstruktor implementálásával. Ennek részleteit most nem taglalom.
Mi hát a lényeg? Azokat az objektumokat, amelyeket MBV módon, más néven kopi-paszta módszerrel akarunk átküldeni a hívott objektumhoz, annak valamelyik fenti módszerrel biztosítsuk a szerializálhatóságát. A legegyszerűbb odabiggyeszteni a [Serializable] attribútumot. Nyilván azonban ez nem mindig gazdaságos.
Gondoljunk a korábbi egymillió elemű tömbös példánkra. Ha a kiszolgáló oldalon futó remoting objektum a tömb jelentős számú elemét fel fogja dolgozni, akkor nincs mese, le kell nyelni, hogy nagyon nagy adattömegek mozognak a metódus hívásakor. De mi van akkor, ha csak véletlenszerű pozíciókon ugyan, de mindig csak például három elemet fog csak kiolvasni a tömbből? Mivel előre nem tudjuk melyeket akarja használni nem tudunk összeállítani egy háromelemű tömböt, és azt átküldeni, hanem kénytelenek vagyunk az egészet átlökni.
Hacsak, ... hacsak nem tudjuk rávenni a kiszolgálóoldalon futó remoting objektumot, hogy nyúljon vissza az ügyfélhez, és olvassa ki a számára szükséges elemeket a tömbből. S bizony rá lehet venni. Ha az átküldendő paraméter a *MarshalByRefObject* nevű osztályból származik, akkor a remoting infrastruktúra teljesen másképp foglalkozik vele. Ebben az esetben a formatterek nem másolják át az objektumot a másik oldalra. Inkább a másik oldalon kiépül egy proxy objektum, amely az ügyfél *MarshalByRefObject* paramétere által hivatkozott objektumot szimulálja. Pont úgy, mint ahogyan a távoli objektum elérésénél megbeszéltük. Ez azt jelenti, hogy ahányszor a távoli objektum hivatkozik valamelyik MarshalByRefObject *(a továbbiakban MBR)* paraméterre, a proxy vissza fog nyúlni az ügyfélbe, hogy kiolvassa a paraméter valamely tagját. Az ügyfél átalakul kiszolgálóvá! Alakítsuk át az

előbbi Kutya objektumot MBR-ré:

```
[Serializable]
class Kutya: MarshalByRefObject {
 private string neve;
 private int kora;
 public int Kor {
 get {
   Console.WriteLine("LekØrdezØtØk a kort.");
   return kora;
   }
 }
}
```

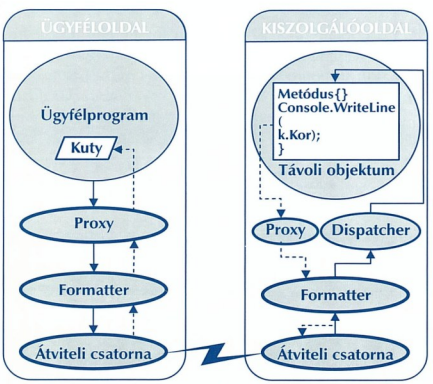
Hogy a későbbi tesztekben láthassuk tényleg visszahívja-e a távoli objektum az ügyfelet a kora tagváltozó eléréséhez írtam egy elérő jellemzőt, amely majd a megfelelő oldalon egy tesztüzenetet ír ki.
A távoli objektum metódusa nézzen így ki:

```
public void Met dus(Kutya k) {
Console.WriteLine(k.Kor);
}
```

A hívás:

```
Kutya kuty = new Kutya();
o.Met dus(kuty) {
```

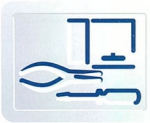
Ha a kutya példány MBR módon viselkedik, akkor a tesztüzenet az ügyféloldalon fog megjelenni, ha MBV módon, akkor pedig kiszolgálóoldalon, hisz ilyenkor tokkal-vonóval átmegy az objektum, nem csak az adatok, hanem a metódusok kódja is.
Mielőtt belevágnánk a gyakorlati implementációba, megpróbálom vizualizálni mi történik az előbbi hívás során:



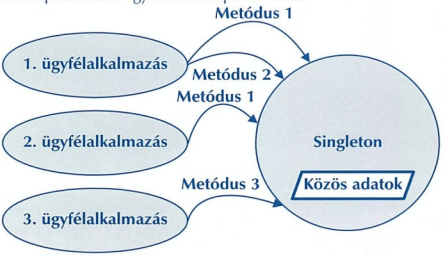
♣ 3. ábra: MBR paraméter elérése

Ha a kutya MBV lenne, akkor a kiszolgáló oldalon is megjelent volna egy másolat-kutya, és a property hívása azon operált volna.
MBV vagy MBR? Ha a paraméterként használt objektum nagy, de csak egy kis részét érjük el a távoli komponensből, és nem kell sokszor visszanyúlni az ügyfélhez *(hálózati körülfordulás!)*, akkor használjunk MBR-t. Párszáz bájtos objektumoknál viszont általában sokkal hatékonyabb tud lenne a MBV.

Remoting objektum fajták

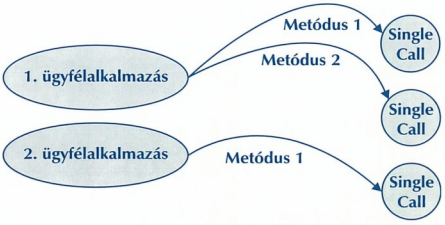


A gyakorlat előtt még feltétlenül meg kell ismerjük, hogy milyen remoting kiszolgáló típusok léteznek. Háromféle típus van, amelyeket aktiválási módjuk szerint két kategóriára osztanak. A Server Activated Objektumokat *(SAO)* a remoting architektúra hozza létre akkor, amikor szükség van rá, nem akkor, amikor egy ügyfélprogram létrehoz rá egy proxyt. Azaz az első távoli metódus hívásakor jönnek létre. Élettartamuk szerint kétféle típusa létezik: a Singleton és a SingleCall objektumok. Azok az objektumok, amelyeket Singletonként jelölünk meg az első ügyfélprogram hívásakor létrejönnek, és egy ideig *(alapban ez 5 perc)* a memóriában maradnak. Ha jön további hívás akár az előbbi, akár más ügyfélalkalmazásokból, ugyanaz az egy objektum fogja őket kiszolgálni. Innen a nevük, hogy egyszerre maximum egy példány létezik belőlük. Koncepcionálisan így lehet elképzelni őket:



♣ 4. ábra: Singleton Server Activated Object

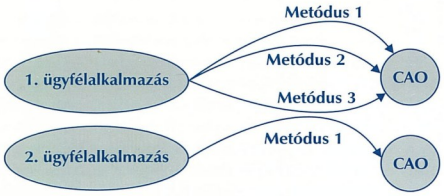
Mivel minden ügyfélprogram ugyanazon a singleton példányon dolgozik, ezért ebben az esetben az ügyfelek látják egymás adatait. Sokszor ez cél, bár ez szinkronizációs problémákat is felvethet. Ezeket az objektumokat nevezik állapotot tároló *(állapotos, stateful)* objektumoknak.
A másik SAO típus a SingleCall remoting objektum. Mint nevében is benne foglaltatik ezen végre lehet hajtani egy metódust, majd ezután az objektum meghal. Ő egy tipikus állapotmentes *(stateless)* objektum, hisz két metódushívás között megsemmisül az egész objektum minden tagváltozójával együtt. Így lehet elképzelni:



♣ 5. ábra: SingleCall Server Activated Object

Minkét típus SAO, azaz egy-egy metódus hívásakor jönnek létre, az élettartamunk kezdetére és végére *(alapban)* az ügyfeleknek nincs ráhatása.
Ezzel szemben a Client Activated Object *(CAO)* típusú remoting objektumokból az ügyfélnek kell létrehozni egy példányt *(amely példány persze a kiszolgálón jön létre)*, és azon végezhet metódushívásokat. Minden egyes ügyfélalkalmazás saját példányt hoz létre, így az ugyanattól az alkalmazástól érkező

egymás utáni hívások ugyanabba az objektumba futnak be. Valahogy így:



♣ 6. ábra: Client Activated Object

A CAO-nak csak ez az egy fajtája van. Állapotot tárol, de ez az állapot ügyfélalkalmazásonként egyedi, nem keverednek az ügyfelek adatai, cserébe egyszerre több példány lebeg a memóriában.

Mikor hal meg egy CAO? Tudományosan megfogalmazva milyen élettartam modellje van a CAO-knak? Kölcsönzős modellt találtak ki nekik *(Lease Based)*. Ez azt jelenti, hogy az objektum létrehozásakor elindul egy óra, amely elkezd visszafelé számolni. Ha lejár az objektumnak annyi. Amikor az objektumot létrehozzuk 5 percről indul a timer. Amikor befut egy metódushívás, a visszaszámlálást visszalökik egy meghatározott idővel *(2 perc)*, de max. a kiinduló ideig. Azaz ha több mint 5 percig nem hívja meg az ügyfél az objektumot *(mert kilépett vagy lefagyott, vagy mert már nincs vele dolga)*, akkor az a szemétgyűjtő martalékává válik. Hasonló a helyzet a Singletonokkal is, csak ott akárhány ügyfélen múlhat az objektum élete.

Ez az idő persze konfigurálható és konfigurálandó, de ez most nem tartozik szorosan a tárgyunkhoz.

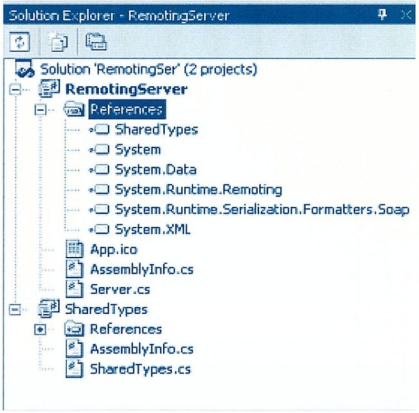
Remoting a gyakorlatban - kiszolgáló

Csapjunk bele, és írjunk meg egy egyszerű remoting ügyfél-kiszolgáló párost. Kezdjük a kiszolgálóval!

A kiszolgáló létrehozásához 4 feladatunk lesz. Először definiálnunk kell azt az objektumot, amelyet szeretnénk távolról elérni. Ez egy class lesz, amelyet a MarshalByRefObject osztályból származtatjuk. Miért is? Emlékezzük vissza, az MBR-eket proxyn keresztül lehet elérni. Ahogyan az 1. ábrán láthattuk az ügyfélprogram proxy segítségével éri el magát a távoli objektumot is, tehát annak MBR-nek kell lenni.

```
public class TavoliObj: MarshalByRefObject { ... }
```

Az ügyfélprogramnak szüksége lesz a TavoliObj metaadataira, azaz a kliens fordításához szükség lesz a TavoliObjot tartalmazó assemblyre. Ennek legegyszerűbb, de nem feltétlen a legjobb megoldása, hogy a TavoliObj-ot egy külön Class Library típusú VS projektbe fordítjuk le. Ennek kimenete egy .dll assembly lesz, erre fog majd hivatkozni az ügyfélprogram is fordításkor. A CLR-el tudatni kell, hogy ezt az objektumot remotingon keresztül szeretnénk publikálni, ehhez kell egy állandóan futó folyamat. Ez lehet egy Windows Szerviz vagy akár az IIS-t is fel lehet kérni erre, de esetünkben ez egy egyszerű konzol alkalmazás lesz, RemotingServer néven:



♣ 7. ábra: Kiszolgálóoldali projektek

A SharedTypes projekt tartalmazza a szerverobjektumot. Látható, hogy a RemotingServer konzolalkalmazásunk vastagon szedett, ő a VS startup projekt, azaz futtatáskor ő indul el. A RemotingServer projekt hivatkozik *(References)* a SharedTypes projektre, hisz szüksége van a TavoliObj lefordított metaadataira.

A továbbiakban a RemotingServer projektben fogunk dolgozni. Második lépésként létre kell hozni azt a csatornát, amin keresztül el lehet érni az objektumunkat:

```
static void Main(string[] args) {
  int port = 2222;
  TcpChannel t = new TcpChannel(port);
```

A TCP csatornánk a 2222-es porton fog figyelni. Ekkor már elkészült a hálózati csatorna, egy socket már ott figyel a proton, csak még a remoting infrastruktúra nem tud róla. Tudassuk vele:

```
ChannelServices.RegisterChannel(t);
```

Már csak egy lépés van hátra, még be kell regisztráltatnunk a TavoliObjektumunkat:

```
RemotingConfiguration.RegisterWellKnownSer-
viceType( typeof(TavoliObj),
"kutya.rem",
WellKnownObjectMode.Singleton);
```

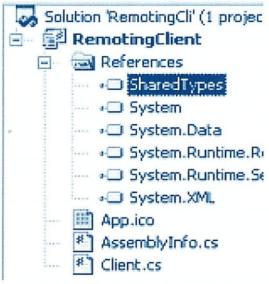
A WellKnownService a SAO egy másik neve *(feltételezem még a béta fázisban menet közben kapott új nevet, csak már nem akarták átnevezni a metódusokat)*. Azaz most az objektumunkat SAO-ként definiáljuk, amely ráadásul Singleton működésű lesz. Ha SingleCall-t szeretnénk, akkor WellKnownObjectMode.SingleCall-t kell megadni harmadik paraméterként. Az első paraméter egy System.Type objektum, amely a Távoli Objektumunkat írja le. Így már érthető miért kell hivatkozunk a SharedTypes projektre, hisz a typeof operátor működéséhez kellenek az adott típus metaadatai.

A második paraméter egy nevet rendel az objektumunkhoz, ezzel lehet majd távolról hivatkozni rá. Nyilván erre azért van szükség, mert egyszerre akárhány remoting objektum csücsül-

het egy kiszolgálón, és mindegyiket egyedi módon kell azonosítani. Hogy itt milyen sztringet adunk meg ránk van bízva, gyakori a valami.rem vagy valami.soap formátum, de bármi lehet.
Ezzel gyakorlatilag felállt a remoting szerverünk. Mivel csak addig hívható az objektumunk amíg az őt regisztráló alkalmazás fut, ezért a konzolalkalmazásunk utolsó sora egy billentyűleütésre vár:

```
Console.ReadLine();
```

Remoting a gyakorlatban - ügyfél
Az ügyfélprogram nagyon hasonló lépésekből épül fel, mint a kiszolgáló. Az ügyfél is egy konzol alkalmazás lesz, amelynek hivatkozni kell a SharedTypes projekt kimenetére, a SharedTypes.dll-re.



♣ 8. ábra: Ügyféloldali projekt

Az ügyfél is létrehozza a csatornáját:

```
static void Main(string[] args) {
  TcpChannel t = new TcpChannel();
```

Látható, hogy most nem adtam meg portot a TcpChannel konstruktorban. Azért nem, mert első körben nem akarjuk, hogy a kiszolgáló visszahívjon az ügyfélbe *(nem használunk MBR paramétert)*, így felesleges az ügyfélnek is hallgatózni.
A következő lépésben tudatjuk a CLR-el, hogy ezt a csatornát szeretnénk majd használni:

```
ChannelServices.RegisterChannel(t);
```

Ez a lépés azonos a kiszolgáló oldalival.
És most jön a mágia, a távoli objektumra mutató proxy létrehozása:

```
TavoliObj o;
o = (TavoliObj)Activator.GetObject(
    typeof(TavoliObj),
    "tcp://localhost:2222/kutya.rem");
```

SAO típusú objektumok létrehozására az Activator.GetObject() metódust használjuk. Első paramétere a létrehozandó objektum típusleírója, a második paramétere pedig egy URL, ez címzi meg a végpontot. Az protokollazonosító *(tcp)* egy kicsit furának tűnhet, de az egyszerűen a TCP csatornát jelenti. Az URL második része a kiszolgáló neve és portja, a harmadik pedig a végpont neve. Emlékezzünk vissza, ezt adtuk meg a távoli objektum regisztrációjakor.
A visszakapott objektum egy proxy osztály, amelyet vissza kell konvertálni a saját távoli objektum típusunkká. A GetObject lefutásakor az ügyféloldali csatorna még nem nyúl át a kiszolgálóhoz, így a kiszolgálóoldali objektum sem aktiválódott még. Akkor fog létrejönni, ha meghívunk rajta egy metódust.

```
MBVKutya mbvLucy = new MBVKutya("MBV Lucy", 3);
o.Metodus2(mbvLucy);
```

Az MBVKutya-t a SharedTypes-ban definiáltuk, mert ezt mindkét oldalnak látni kell:

```
[Serializable]
public class MBVKutya {
  private string neve;
  private int kora;

  public MBVKutya(string neve, int kora) {
    Console.WriteLine("Kutya aktivÆl dott");
    this.neve = neve;
    this.kora = kora;
  }

  public int Kor {
    get {
     Console.WriteLine("LekØrdezØtØk a koromat.");
     return kora;
    }
  }
}
```

Gondolom világos miért az MBV előtag: ez egy Serializable osztály, ami nem a MarshalByReferenceObjectből származik, így MBV módon fog közlekedni.
A kiszolgálóoldali objektum törzse ekképpen néz ki:

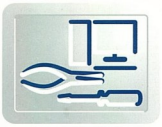
```
public class TavoliObj: MarshalByRefObject {
  public TavoliObj() {
    Console.WriteLine("TavoliObj lØtrej tt.");
  }

  ~TavoliObj() {
    Console.WriteLine("TavoliObj megsemmis lt.");
  }

  public void Metodus2(MBVKutya kuty) {
    Console.WriteLine(kuty.Kor);
  }
}
```

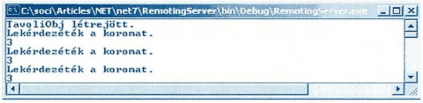
A Metodus2 lekérdezi a kapott MBVKutya Kor jellemzőjét. Az osztálynak írtam konstruktort is, így látható lesz mikor hoz létre belőle a CLR példány*(oka)*t, valamint kapott egy destrukort is, így látni fogjuk a szemétgyűjtő munkáját is, illetve ellenőrizhetjük az objektumok élettartam kérdéseit is.
Futtassuk le a példánkat! Ügyféloldalon láthatjuk, hogy az MBVKutyából létrejött egy példány:

Kiszolgálóoldalon a metódushívás hatására létrejön a Singleton SAO, és lefut a MBVKutya jellemző mögötti kód.



Hogy miért kiszolgálóoldalon fut le jellemző lekérdezése remélem teljesen egyértelmű: az MBV paraméter másolata létrejött kiszolgálóoldalon, így a jellemző mögötti kód is ott fut le, és az a másolat adatait éri el.

Ha 5 percen további hívásokat eszközölünk, akkor azok ugyanarra az objektumra érkeznek be, ezért Singleton a típusa:



Ha viszont több mint 5 percig nem nyúlunk a Singletonunkhoz, akkor a remoting infrastruktúra létrehoz egy új példányt, így a következő hívások már egy új példányhoz fognak becsapni:

De miért nem látszik, hogy az első példány meghalt? Ennek oka a szemétgyűjtő működésében keresendő. Bár már a korábbi számokban kifejtettem most újra megismétlem: a .NET Framework memóriamodelljében a nem hivatkozott objektumok felszabadítása időben nem determinisztikus. A szemétgyűjtő akkor takarít ha már nincs hely létrehozni új objektumokat. Az előbbi példában a CLR elengedte az első Singleton objektum példányt, azaz eldobta a rá mutató referenciáját, és létrehozott egy második példányt. De .NET-ben nincs mód explicit objektum felszabadításra, így egyszerűen tovább lebeg a memóriában az első *(már nem használt)* példány.

Mikor takarodik el onnan? Majd ha elindul a szemétgyűjtő *(Garbage Collector, GC)*. Egyszer csak. Nem érdekes mikor, csak emlékezzünk rá, hogy így működik. Ezért ne nyissunk meg például adatbáziskapcsolatot egy .NET-es objektum *(mindegy hogy remote vagy nem)* konstruktorában, mert a destruktor lehet, hogy nagyon sokára hívódik meg, így akár több ezer adatbáziskapcsolatot is felemészthetnek a lebegő objektumok. De vissza a remotingra. Regisztráljuk most be az objektumunkat SingleCallként!

```
RemotingConfiguration.RegisterWellKnownServiceType
( typeof(TavoliObj),
  "kutya.rem",
  WellKnownObjectMode.SingleCall);
```

Az ügyfélprogram változatlan maradt. Nézzük meg a kiszolgáló konzolját:

Háromszor hívtam meg a Metódus2-t, és látható, hogy mindhármat külön objektum szolgálta ki. Plusz egy létrejön az első hívás után is, ennek okáról még az Advanced .NET Remoting könyv is **[1]** hallgat.

Zavarjuk meg egy kicsit a kiszolgálót! Hívjuk meg ezerszer ugyanazt a metódust:

```
for(int i=0; i<1000; i++) {
  o.Metodus2(mbvLucy);
}
```

Látható, hogy ezt már nem bírja idegekkel a tisztasági vállalat sem, és elkezdi kidobálni az elhasznált, lebegő SingleCall objektumokat *(9. ábra)*.

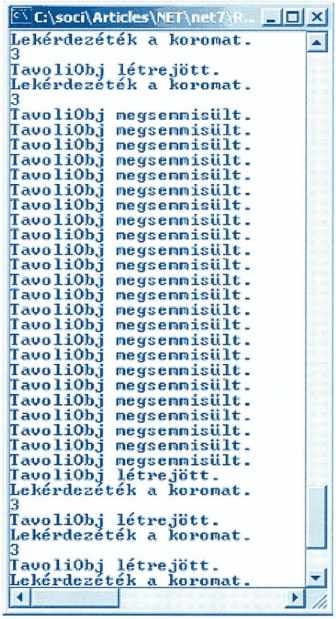
MBR paraméterek

Tekintsük meg a másik kutyánkat:

```
[Serializable]
public class MBRKutya: MarshalByRefObject {
...
}
```

Tartalmát tekintve azonos az MBVKutyával, csak ez a MarshalByRefObjectből származik, így ennek megfelelően fog viselkedni.

A kiszolgálóobjektum másik metódusa ezen a típuson ügyködik:



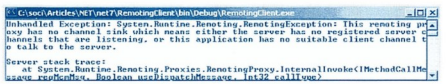
♣ *9. ábra: A szemétgyűjtő kidobálja a már nem használt SingleCall objektumokat*

```
public class TavoliObj: MarshalByRefObject {
  public void Metodus1(MBRKutya kuty) {
    Console.WriteLine(kuty.Kor);
...
```

Az ügyfél így próbálkozik:

```
MBRKutya mbrBobi = new MBRKutya(
  "MBR Bobi", 2);
o.Metodus1(mbrBobi);
```

Ám nagy bukás a vége:



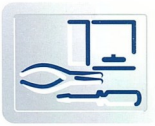
A kivétel szövege egy kicsit félreérthető. Arról papol, hogy a szervernek nincs hallgatózó csatornája. Arról van szó, hogy a távoli objektumot futtató kiszolgáló az MBR paraméter jellemzőjének végrehajtásához vissza akar csatlakozni az ügyfélhez. Ehhez az ügyfélnek szerverként kell viselkedni, azaz listen-ölni *(figyelni)* kell valamilyen porton, a tényleges szerverhez hasonlóan. Ehhez az ügyfélprogram elejét kell módosítanunk úgy, hogy a TcpChannel-nek adunk egy port paramétert:

```
TcpChannel t = new TcpChannel(3333);
```

Ettől kezdve a kiszolgáló vissza tud csatlakozni az ügyfélre a 3333-as TCP porton, így meg tudja hívni a „Kor" nevű jellemző mögötti kódot.
Nézzük meg a hívás után a kiszolgáló ablakát:

Sehol sincs arra utaló nyom, hogy a Kor jellemzőt lekérdezték volna! Na, de az ügyfélnél:

Ez szép! A kiszolgáló visszahívott minket. Ez a rendszergazdák réme, ha a kiszolgáló tűzfalon kívül van, az ügyfél pedig a vállalati hálózaton belül csücsül. Ebben az esetben ugyanis az ügyfél kezdeményez egy hálózati kapcsolatot kifelé a tűzfalon keresztül, ami normális. Ám a kiszolgáló illetlen módon vissza akar bújni a tűzfalon keresztül az ügyfélhez, és ettől garantáltan idegbajt kapnak a tűzfalgazdák. Ezért utálják az FTP-t is, mert az is visszafelé építi ki az adatcsatornáját. FTP-hez az okos tűzfalak *(stateful inspection technológiát használók, mint a Checkpoint vagy az ISA)* képesek a kifelé irányuló forgalomban felfedezni azt, hogy majd jönni fog egy visszirányú csatornakérelem, így dinamikusan, abban a pillanatban ki tudnak nyitni neki egy portot, csak az adott forrás és célcím között. Azonban a mai tűzfalaknak beleértve az ISA-t is közük nincs a .NET remoting protokolljához, így az FTP-vel ellentétben ez nem fog menni.
Ez azért tud fejfájós lenni, mert a remoting remekül képes átjátszani például az event-öket is. Így például egy Singleton kiszolgáló feldobhat egy eseményt, amit akárhány ügyfél megkaphat *(pl. Chat program)*. Ebben az esetben nyilvánvalóan a szerver vissza kell, hogy szóljon az ügyfeleknek, és jön a tűzfalas probléma. Ezt csak úgy lehetne megúszni, ha az ügyfél folyamatosan fenntartaná a transzport *(TCP)* csatornáját a kiszolgáló felé, és az ügyféloldali irányból kiépült csatornán a kiszolgáló remekül dumálhatna visszafelé. De a remoting minden egyes kérésnél újranyitja a csatornát, majd a metódushívás végén lezárja, ezért kell a kiszolgálónak visszacsatlakozni.



CAO-k

A CAO-k egy kicsit másképp működnek, mint a SAO-k. A csatornaregisztrációig minden azonos a két típus esetén. Ám ezután a kiszolgáló így néz ki:

```
RemotingConfiguration.RegisterActivatedServiceType(
typeof(TavoliObj) );
```

Most nincs a végpontnak azonosítója, csak meg kell adni az ügyféloldalról aktiválandó típust, és ezúttal a RegisterActivatedServiceType metódust kell használni. Az ActivatedType a Client Activated rövidített neve. A végpontnak ezért nincs azonosítója, mert minden egyes ügyfél kap egy egyedi azonosítót az objektum létrehozásakor, így lehetséges az, hogy minden egyes ügyfél egy saját kiszolgálópéldányon hajthatja végre a hívásait.
Mivel a CAO-k esetén pont a személyre szabott állapottárolása az érdekes, ezért a teszteléshez egészítsük ki a kiszolgáló objektumot:

```
public class TavoliObj: MarshalByRefObject {
  private MBVKutya k;
  public MBVKutya GetKutya() {
    return k;
  }
  public void SetKutya(MBVKutya k) {
    this.k = k;
  }
...
```

Azaz viszünk némi állapottárolást a kiszolgálóba.
Ügyféloldalon is létre kell hozni és beregisztrálni a csatornát, pont, mint a SAO-knál. Az aktiválás viszont másképp néz ki:

```
TavoliObj o;
RemotingConfiguration.RegisterActivatedClientType(
  typeof(TavoliObj),
  "tcp://localhost:2222");

o = new TavoliObj();
```

A RegisterActivatedClientType hívással tudatjuk a CLR-rel, hogy a TavoliObj típusú CAO-nkat mely csatornán és kiszolgálón lehet elérni. Ezek után a new operátor működése átalakul. Ha egy már beregisztrált CAO-ból szeretnénk példány létrehozni, akkor nem lokálisan jön létre az objektumunk, hanem a kiszolgálón! Mágia.
Teszteljük le az objektumunk állapottárolását:

```
MBVKutya mbvLucy = new MBVKutya(
  "MBV Tacsi", 6);
o.SetKutya(mbvLucy);
MBVKutya kk = o.GetKutya();
Console.WriteLine(kk.Kor);
```

Az ügyfél kimenetéből látszik, hogy sikerült visszakapni a korábban eltárolt objektum referenciát:

Hogy nem történt csalás, azaz, hogy tényleg eljutottunk a kiszolgálóig:

Persze több, különböző paraméterekkel indított ügyféllel lehet igazán megvizsgálni, hogy tényleg nem egy Singleton objektumról van szó, hanem egy ügyfelenként egyedi adatokat tároló CAO-ról.

Konfiguráció

A .NET keretrendszer nagyon sok szinten konfigurálható. Nem kivétel ez alól a remoting sem. Mind a csatornák, mind az objektumok regisztrációját konfigurációs fájlokban is leírhatjuk, aminek nagy előnye, hogy például a kiszolgáló nevének megváltozás miatt nem kell újrafordítani az alkalmazást, elég csak a konfigurációs állományokat átírni.
Nézzük meg először a kiszolgáló oldalt. A csatorna és kiszolgáló objektum konfigurációja rettenetesen egyszerű:

```
RemotingConfiguration.Configure(
  "RemotingServer.exe.config");
```

Ennyi az egész, a kiszolgálót talpra állító kód 1 azaz 1 sorból áll! Persze ehhez kell egy kulturáltan kitöltött konfigurációs fájl, amit én most az alkalmazás konfigurációs fájlba raktam, de tetszőleges szövegfájlba is lehetne. Egy tipp a VS.NET használóknak. Az alkalmazás konfigurációs fájlnak az alkalmazás neve kiterjesztéssel plusz .confignak kell lennie. A VS a C# konzol projekteket a bin/Debug vagy bin/Release mappába fordítja le. Azaz a konfigurációs állománynak itt kell lakni. Azonban elég kényelmetlen lenne mindig két példányt módosítani a kétféle *(Debug vagy Release)* verzió miatt. Ezt megkönnyítendő hozzunk létre egy „app.config" *(szó szerint, nem alkalmazásnév.config)* állományt a projektben, és a VS minden egyes Build esetén bemásolja a tartalmát megfelelő fájlnévvel az aktuális kimeneti könyvtárba.
Lássuk a kiszolgálóoldali konfigfájl tartalmát *(CAO esetére)*:

```
<configuration>
 <system.runtime.remoting>
  <application name="TesztSzerver">
   <service>
    <activated
     →type="SharedTypes.TavoliObj, SharedTypes">
    </activated>
   </service>

    <channels>

    <channel port="2222"
    →type="System.Runtime.Remoting.Channels.Tcp.
    →TcpChannel, System.Runtime.Remoting,
    →Version=1.0.3300.0, Culture=neutral,
    →PublicKeyToken=b77a5c561934e089" />

    <channel port="2223"
    →type="System.Runtime.Remoting.Channels.Http.
    →HttpChannel, System.Runtime.Remoting,
    →Version=1.0.3300.0, Culture=neutral,
    →PublicKeyToken=b77a5c561934e089" />

    </channels>
  </application>
 </system.runtime.remoting>
</configuration>
```

Az <application> elem name attribútuma a GUI alapú adminisztrációs programok kedvéért ad egy nevet a konfigurációnknak.
A <service> szekció írja le az alkalmazás által publikált típusokat és azok aktiválási módjait. Az <activated> ismét a CAO-ra utal. A type attribútum írja le a kiszolgálóobjektumot. Az első része az típus teljes, azaz a névtérrel kiegészített neve, a második rész pedig az őt tartalmazó assembly neve. Esetünkben a TavoliObj a SharedTypes nevű assembly lakója, ami a SharedTypes.dll-ben lakik. Fontos, hogy az .EXE alkalmazásunk részére valami módon elérhető legyen a publikálandó típust tartalmazó assembly, esetünkben a projektek közötti Referencia miatt a VS minden egyes fordítás után a SharedTypes.dll-t átmásolja a RemotingServer.exe mellé.
A <channels> szekció írja le azokat a csatornákat, ahol a kiszolgálónk várja az aktiválási kérelmeket. Láthatóan le kell írni a csatorna típusát és a típust tartalmazó assembly nevét, pont úgy, mint az <activated> elemnél. Az assembly név most teljesen kifejtett, mert a System.Runtime.Remoting assembly egy strong name-el ellátott *(digitálisan aláírt)* assembly, így a teljes nevével kell rá hivatkozni. Ezt a hosszú nevet a GACUTIL /L >out.txt parancssori hívásával a legegyszerűbb kiolvasni.
Ha már ilyen egyszerű a konfiguráció egyből beregisztráltattam egy HTTP csatornát is a 2223-as porton, így az ügyfeleken múlik, hogy melyiken aktiválják az objektumot. Ha SAO-ként szeretnénk használni a TavoliObj-t, akkor a <service> szekció így változik:

```
<wellknown mode="Singleton"
type="SharedTypes.TavoliObj, SharedTypes"
objectUri="kutya.rem" />
```

Azaz az <activated> helyett <wellknown> elemre van szükségünk, és ebben az esetben kell egy azonosító is az objektumunkhoz *(kutya.rem)*.
Ha SingleCall élettartam modellre van szükségünk, akkor a mode attribútumot erre kell kicserélni.
Szemmel láthatóan nincs itt semmi mágia, egyszerűen azokat az adatokat, amelyeket korábban programozottan adtunk meg most egy xml fájlban gyűjtöttük össze.
Az ügyféloldali konfiguráció hasonlóan egyszerű:

```
<configuration>
 <system.runtime.remoting>
  <application name="TesztRemotingUgyfel">
   <client displayName="RemotingUgyfelPelda"
    →url="http://localhost:2223">
    <activated
    →type="SharedTypes.TavoliObj, SharedTypes"
    />
   </client>
   <channels>
    <channel port="0"
     →type="System.Runtime.Remoting.Channels.Http
     →HttpChannel, System.Runtime.Remoting,
     →Version=1.0.3300.0, Culture=neutral,
     →PublicKeyToken=b77a5c561934e089" />
   </channels>
  </application>
 </system.runtime.remoting>
</configuration>
```

A csatorna konfigurációban csak annyi a különbség, hogy a csatornánk most nem hallgatózik ügyféloldalon *(port="0")*, ami azt jelenti, hogy az MBR paraméterek nem működnének. Persze ha arra van igény, egyszerűen meg kell adni egy valós portszámot. Most HTTP protokollon megyünk be a kiszolgálóhoz, a 2223-as porton. Emlékezzünk vissza, ott hallgatózik a HTTP csatorna. A <service> elem helyett most <client> áll, és itt kell megadni a CAO URL-jét is.
SAO-k esetén a <client> szekció egy kicsit másképp néz ki:

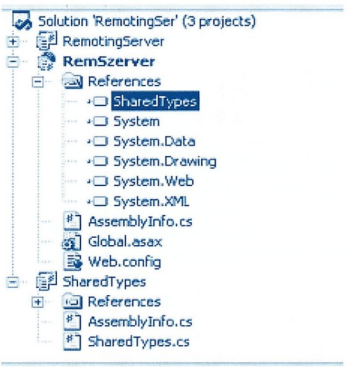
```
<client displayName="RemotingUgyfelSAO">
 <wellknown
  →type="SharedTypes.TavoliObj, SharedTypes"
  →url="http://localhost:2223/kutya.rem"/>
</client>
```

Azaz az URL-t most a <wellknown> *(SAO)* elem attribútumaként kell megadni. A SAO-k esetén a konfigurációs állomány további előnye, hogy a nehézkes Activator.GetObject helyett egyszerűen a new operátorral hozhatjuk létre a távoli típust *(mint a CAO-kat)*.
Láthatjuk, hogy mindkét oldal immáron egy-két sorosra egyszerűsödött le, az összes piszkos munkát elvégzi helyettünk a remoting infrastruktúra. De van még egy pont, ahol egyszerűsíthetünk a munkánkon. Emlékezzünk arra, hogy a remoting objektum csak addig áll az ügyfelek rendelkezésére, amíg az őt regisztráló folyamat él. Ez most egy konzolalkalmazás volt, a gyakorlatban praktikusan egy Szerviz lenne. De még ezt is meg lehet spórolni. Valaki folyamatosan ott figyel a gépünk legalább egy portján... Igen, ő az IIS, a webkiszolgáló. Miért ne lehetne ő a publikálás atyja?

Hosztolás IIS-ben

Mivel az IIS egy HTTP kiszolgáló, ezért nyilván egy benne hosztolt remoting objektumot csak HTTP csatornán keresztül lehet elérni, TCP-n nem. Ha szükséges a TCP nyers teljesítménye, akkor meg kell írni a felhúzó alkalmazást.
Hogyan tudathatjuk az IIS-el, hogy valamely típusunkat szeretnénk általa publikálni? Természetesen konfigurációs fájlokkal. Az IIS konfigfájlok neve web.config, és bármelyik webalkalmazás gyökerében sőt bármelyik alkönyvtárban elhelyezhetők.
A webalkalmazásoknak van egy központi állománya is, ez a global.asax. Ebben ráakaszkodhatunk olyan eseményekre, mint például elindult a webalkalmazás. Ezt kihasználhatjuk a remoting további konfigurálására.
A teszt kedvéért készítsünk egy web projektet a http://localhost/RemSzerver címre. Ebben nem lesznek weboldalak *(de persze lehetnének)*, csak a remoting típusainkat fogjuk itt publikálni. A VS által generált felesleges állományokat kitöröljük, csak a lényeg maradjon meg, ahogyan az a 10. ábrán is látható. A Referencesben most is hivatkozunk a SharedTypes projektre, így annak output assemblyjét a VS bemásolja a webprojektünk bin könyvtárába, ezért a remoting infrastruktúra el tudja érni a típusdefiníciókat.
Nézzük meg hogyan kell megírni egy Singleton SAO publikálására szolgáló web.config-ot:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
 <system.runtime.remoting>
  <application>
   <service>
   <wellknown mode="Singleton"
    →type="SharedTypes.TavoliObj, SharedTypes"
    →objectUri="kutya.rem" />
   </service>
  </application>
 </system.runtime.remoting>
</configuration>
```



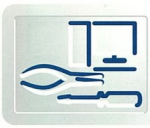
♣ 10. ábra: A remoting objektumunkat hosztoló webprojekt

A konfigurálás láthatóan megegyezik a korábbi példáéval, csak a csatornát nem kell definiálni. Ennek az az oka, hogy most az IIS fog figyelni HTTP csatornaként, ezért értelmetlen lenne megadni ugyanezt még egyszer a web.config-ban.
Egy másik apró változás, hogy az <application> elemben nem kell és nem is lehet nevet adni az alkalmazásnak, mert a webalkalmazás neve lesz az alkalmazás neve.
Nagyon fontos viszont, hogy a végpont nevének kiterjesztése mindenképpen .rem vagy .soap legyen, ugyanis az IIS-ben erre a két kiterjesztésre van bekonfigurálva a remoting kiszolgáló. Ezért a kutya.rem.
Ennyi az egész! Készen van a remoting kiszolgálónk. Elég elhelyezni a megosztott assemblyt a webalkalmazás bin könyvtárába *(VS megtette)*, és megfelelően kitölteni a web.config-ot, és máris működik a kiszolgáló. Ez azért értékelendő!
Az ügyfelet csak át kell konfigurálni az új url használatára:

```
<wellknown type="SharedTypes.TavoliObj,
→SharedTypes"
→url="http://localhost/RemSzerver/kutya.rem"/>
```

A végpont most a 80-as porton érhető el, hisz ott figyel a webszerver. Ha ez nem megfelelő, akkor az IIS-ben létre kell hozni egy új website-ot a megfelelő porton, és oda elhelyezni a publikálandó objektumot.
Még néhány kiegészítés. A HTTP és a TCP csatornához tartozó típus és assembly nevek már előre benne vannak a központi machine.config állományban, ezért sokkal egyszerűbben is hivatkozhatunk rájuk:

```
<channel ref="http" ...>
```



A csatornáknak további jellemzői is konfigurálhatóak. Például a http csatornának megmondhatjuk *(ügyféloldalon)*, hogy ne az alapértelmezett SOAP, hanem bináris formattert használjon:

```
<channel ref="http">
  <clientProviders>
    <formatter ref="binary" />
  </clientProviders>
</channel>
```

Ez nyilvánvalóan jót fog tenni az átviteli teljesítménynek, de nézzük csak, mennyire?

Teljesítménytesztek

Mi befolyásolja a távoli hívások sebességét? Természetesen alapvetően a hálózati protokoll és a formatter. A másik erősen befolyásoló paraméter az átvitt objektumok mérete és szerkezete. A méret és a teljesítmény közötti összefüggés triviális: minél nagyobb objektumokat kell átcipelni a hálózaton annál több ideig dolgozik az átviteli csatorna. Ezen még az alacsonyszintű TCP csatorna se tud segíteni.

A paraméterként használt objektumok szerkezetétől való teljesítményfüggés vizsgálatához már ismerni kell hogyan dolgoznak a formatterek. A bináris formázó kb. akkora adatcsomagot generál mint az objektum memóriabeli képe. A SOAP formázó által előállított xml csomag viszont lényegesen nagyobb is lehet, mert kb. minden tagváltozó *(mező)* neve is megjelenik egy xml elemként a csomagban, valamint a SOAP boríték is jó pár bájtból áll. A hosszú nevű tagváltozók okozta költség várhatóan akkor lesz szembetűnő, ha az objektum sok tagváltozót tartalmaz, de a teljes mérete kicsi *(végig MBV-ről beszélünk)*. A boríték súlya is inkább kis objektumoknál fog előtérbe kerülni.

A tesztünkhöz a következő objektumokat fogjuk használni.

Név	Jellemző
A	Kis objektum kevés mezővel
B	Kis objektum sok mezővel
C	Nagy objektum kevés mezővel
D	Nagy objektum sok mezővel

Az objektumok definíciója:

```
/// <summary>
/// Kis objektum kevős mezıvel.
/// </summary>
[Serializable]
public class A {
  public int Labak;
  public int Kor;
}

/// <summary>
/// Nagy objektum kevős mezıvel.
/// </summary>
[Serializable]
public class C {
  public string Neve;
  public int Kor;
}
```

```
/// <summary>
/// Kis objektum sok mezıvel.
/// </summary>
[Serializable]
public class B {
  public int Labak;
  public int Kor;
  public byte BaratokSzama;
  public short Szine;
  public int FarkaSzama;
  public int Labmerete;
  public int Fejmerete;
  public double Marmagassaga;
}

/// <summary>
/// Nagy objektum sok mezıvel.
/// </summary>
[Serializable]
public class D {
  public int Labak;
  public int Kor;
  public byte BaratokSzama;
  public short Szine;
  public int FarkaSzama;
  public int Labmerete;
  public int Fejmerete;
  public double Marmagassaga;
  public string Neve;
}
```

A nagy objektumok terhelését a 100,000 bájt méretűre feltöltött Neve mező fogja okozni.

A távoli objektum definíciója:

```
public class MeasureServer: MarshalByRefObject {
  public void Set(object x) {}
}
```

Azaz csak odafelé fognak áramlani paraméterek, mert a paraméterátadás in típusú, és bár az egyszerűbb kezelés érdekében a formális paraméter object típusú, a valódi paraméterek a fenti négy típusból kerülnek majd ki. Minden mérésben 100-szor hívjuk meg a metódust, és minden sorozat előtt egy távoli objektumpéldányt mérés nélkül hozunk létre, hisz egy Singleton esetén az egyetlen példány létrejötte sokkal tovább tart, mint utána a hívások. A mérésekben a Release kódokat használtam.

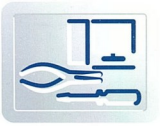
A MeasureServer Singletonként van beélesztve az IIS alá. Emellett a MeasureServer típust kipublikáljuk még a korábbi konzol alkalmazásunkból is HTTPn és TCP-n is, szintén Singletonként. Ez a HTTP nem lesz azonos az IIS általi HTTP kiszolgálóval, két különböző implementációról van szó. Az alapvető különbség a kettő között, hogy az IIS-es verzióban az IIS *(inetinfo.exe)* és az ASP.NET futtató *(aspnet_wp.exe)* közötti kommunikáció nagyon jelentős időt vihet el a hívások során. Ezen majd a .NET Server fog segíteni *(IIS6)*.

S ha már az IIS tud titkosítani *(HTTPS)*, miért nem mérnénk ki annak is a költségét? Ehhez némi trükközésre van szükség, mert a titkosításhoz szükséges tanúsítványt *(Certificate)* magam generáltam, így a tanúsítványkiadó *(a saját Certificate Serverem)* nincs benne a megbízható kiadók listájában, így a HTTP ügyfél élből nem hajlandó vele kommunikálni. Ez persze orvosolható egy gyengített Certificate Policy felállításával, ennek részletei a melléket forráskódban láthatóak.

Ezek alapján 8 mérést végzünk el, mindegyikben négyféle objektummal. Az ügyfél és a kiszolgáló ugyanazon a gépen futott, emiatt a mérési eredmények némileg eltérhetnek a valódi többgépes megoldásétól.

Lássuk az eredményeket:

IISHTTP-SOAP	
A	3.234s
B	3.274s
C	10.14s
D	10.214s
IISHTTP-Binary	
A	3.164s
B	3.284s
C	9.864s
D	10.525s
IISHTTPS-SOAP	
A	5.237s
B	4.296s
C	13.789s
D	14.0s
IISHTTPS-Binary	
A	4.216s
B	4.316s
C	14.90s
D	14.430s
HTTP-SOAP	
A	2.153s
B	2.253s
C	8.321s
D	8.311s
HTTP-Binary	
A	2.413s
B	2.383s
C	8.362s
D	8.562s
TCP-SOAP	
A	1.522s
B	1.522s
C	6.339s
D	6.389s
TCP-Binary	
A	1.442s
B	1.542s
C	6.449s
D	6.629s



Vonjunk le néhány tanulságot. Mind kis, mind nagy objektumok esetén a TCP-Bináros páros szinte verhetetlen, de a TCP-SOAP-nak sem kell szégyenkezni. Lassú vonalon azért a Bináris jelentősen gyorsabb lett volna *(tesék otthon kipróbálni)*. Az IIS-ben kontra natívan hosztolt HTTP esetén az IIS kb. másfélszer lassabb. Viszont sok párhuzamos ügyfél esetén lehet, hogy hatékonyabb lenne, mint a társa. A mérés alapján az IISnél szinte mindegy, hogy milyen formátumot használunk *(de ezt azért nem merném általánosítani)*. Ami még nagyon érdekes, hogy a titkosítás alig lassít az eljáráshívásokon, még nagy paraméterek használata mellett is. De kár lenne nem kihasználni!

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

A cikkben szereplő URL-ek:

[1]: Ingo Rammer: Advanced .NET Remoting, Apress, 2002 *(könyv)*
[2]: Letölthető példakódok
http://technet.netacademia.net/download/dotnet

Az Exchange 2000 Server és a Web Storage System

Úgy gondolom, mindenképp érdemes néhány szót szólni a WSS sémáról is, amely adataink struktúráját, felépítését határozza meg.

Property, content class és séma

A Web Storage System egy lazán strukturált adatbázis, nincsenek olyan szigorú megkörések, mint például az SQL esetén. Valamilyen szintű szervezettségre azonban szükség van, hiszen a teljes szabadság többnyire csak káoszt eredményez, hatékony megoldásokat nem igazán…

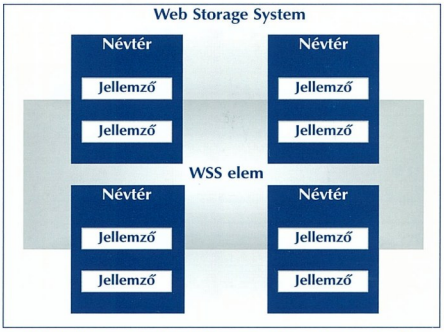
A propertyk az adatainkhoz köthető jellemzők, melyek leírják azt. Rengeteg előre definiált ilyen jellemző van a WSS-ben, de saját jellemzőket is vehetünk fel *(lásd később)*.

A WSS-ben minden adatnak van egy content classa, amely azt mutatja meg, hogy az adatunk milyen típusú, milyen propertyk vannak hozzárendelve alapértelmezettként. Beépített és saját content classokról is egyaránt beszélhetünk.

Hétköznapi példákat is nagyon könnyen találhatunk ezen fogalmak tisztázására. Bármire gondolunk a bennünket körülvevő világból, azt is besoroljuk valamilyen osztályba, és gondolatunkban *(akár tudatosan, akár ösztönösen)* egy köteg jellemzőt rendelünk hozzá. Gondoljunk például egy autóra. Ezzel máris megadtuk a kívánt objektum „content classát": autó. Gondolatainkban azonban az autó osztály egy konkrét példánya, például a férjünk/feleségünk tűzpiros Ferrarija jelenik meg, ezzel máris megvannak az autó jellemzői is: mi a típusa, hány ajtaja van, milyen színű, hány személyes, milyen váltó van benne, van-e benne légzsák az anyósülésnél, milyen extrákkal van felszerelve stb.

Mindennapjainkban ilyen és ehhez hasonló sablonokban, sémákban gondolkodunk.

A Web Storage Systemben a séma adatdefiníciós eszköz, amely előre definiált jellemzők halmaza, és a WSS elemeit *(mappák, file-ok, stb.)*, azok viselkedését határozza meg. Ezek a jellemzők általában névterekbe *(namespace)* vannak rendezve, és saját jellemző felvételénél is ajánlott ez a fajta szervezettség. A különböző névterek természetes csoportokba szervezik a jellemzőket, és segítségükkel elkerülhetőek az ütközések is. Elemeink általában nem egyetlen névtérből kapnak jellemzőket, minden elemhez különböző jellemzőhalmaz tartozhat. Egy-egy névtérből azonban nem kötelező az összes jellemzőt felhasználni. Ez a rugalmasság biztosítja, hogy a mappák alkalmasak ilyen sokféle, változatos adatok tárolására és szervezésére. Az alábbi ábra a jellemzők, névterek, elemek és sémák közötti kapcsolatot mutatja:

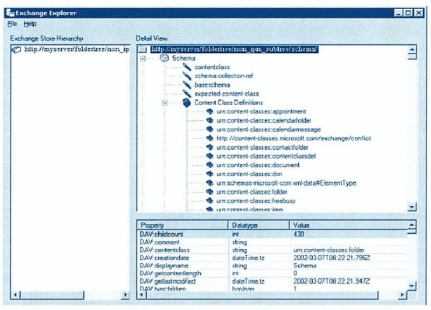


♣ ***A Web Storage System jellemzőinek, névtereinek, elemeinek és sémáinak kapcsolata***

A Web Storage System alapsémája a jellemzők rendkívül változatos sokaságát kínálja, így az esetek többségében alkalmas saját alkalmazásunk adatainak felépítésére is. Azonban arra is lehetőségünk nyílik, hogy a sémát saját jellemzőinkkel bővítsük, mi több, saját sémát is létrehozhatunk.

Séma bővítése, létrehozása

Lássuk, hogyan épülnek fel a sémák! A WSS alapsémája *(baseschema)* a Foldertree/non_ipm_subtree/schema/ helyen található. Ez egy rejtett mappa, és XML adatokat tartalmaz. Ha megnyitjuk az Exchange Explorerrel, az alábbihoz hasonló kép tárul a szemünk elé:



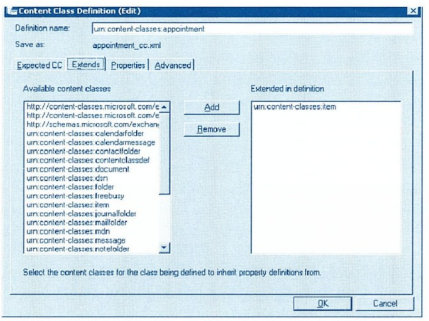
♣ ***A WSS alapséma***

Láthatjuk, hogy sok-sok content-class és property definíció található itt. Ha az item-eket megnézzük, olyan XML fájlokat találhatunk, amelyek ezeket a definíciókat tárolják.
Nézzünk meg először egy jellemzőt közelebbről! A Property Definitions ágat kifejtve jobb egérgombbal kattintsunk egy jellemzőre, majd Edit Property. A kinyíló ablak ismerős lehet már, hiszen csaknem ugyanaz, mint amikor új jellemzőt veszünk fel *(lásd egyik korábbi cikkemben)*. Ez az ablak tehát a propertyt eltároló XML fájlból veszi az adatokat, amelynek felépítése az alábbihoz hasonló:

```
<?xml version="1.0" ?>
<Schema name="ExchangeSchema"
→ xmlns="urn:schemas-microsoft-com:xml-data"
→ xmlns:d="DAV:" xmlns:cc="urn:content-classes:"
→ xmlns:dt="urn:schemas-microsoft-com:datatypes"
→ xmlns:s="urn:schemas-microsoft-com:exch-data:"
→ xmlns:h="urn:schemas:mailheader:">

   <ElementType name="h:subject" dt:type="string"
→ s:ismultivalued="0"
→ d:contentclass="cc:propertydef" s:isindexed="0"
→ s:isreadonly="0" s:isrequired="0"
→ s:isvisible="1" s:synchronize="0" />
</Schema>
```

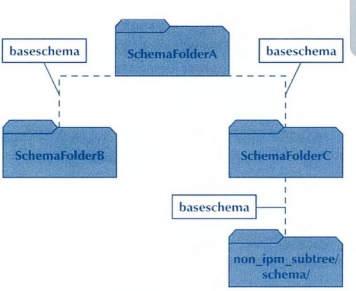
A content classok belsejébe hasonló módon tekinthetünk. Az alább látható ablak felső részében a CC és az őt tároló XML file nevét láthatjuk. Alatta négy ablakrész látható. Az Extends részen azt adhatjuk meg, hogy melyik CC-ból származik, honnan örökli alaptulajdonságait az éppen aktuális. Az alábbi ábrán azt láthatjuk, hogy az Appointment az Item-re épít:



♣ *WSS Content Class*

A Properties fülön azt állíthatjuk be, hogy az alap content class jellemzőin kívül milyen, korábban már definiált jellemzőket adunk az új CC-hoz.
Az alapséma módosítása tehát úgy történik, hogy a benne definiált property-ket és/vagy content classokat módosítjuk. Ez járható út, azonban nem szerencsés, hiszen ezt a sémát több alkalmazás is használhatja, amik esetleg nem tolerálják a változásokat.

Ezért sokkal szerencsésebb, ha saját sémát *(sémákat)* hozunk létre. A sémák hierarchikusan szervezhetőek, így például az alábbi ábrán látható esetben a SchemaFolderA felhasználja a SchemaFolderB és a SchemaFolderC sémát, míg ez utóbbi a WSS alapsémájára épít.

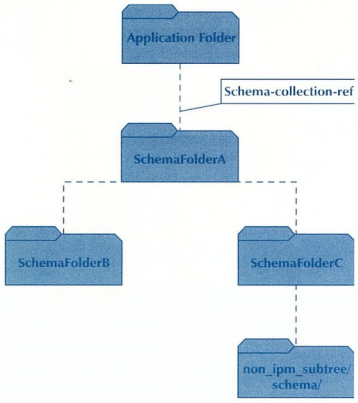


♣ *A Web Storage System sémahierarchiája*

Hogyan érhető el mindez? Először is hozzunk létre az alkalmazásunkhoz egy sémamappát *(melyet később érdemes rejtetté tenni)*, praktikusan a /Foldetree/MyApplications/Schema/ helyen. Az Exchange Explorer Detail View ablakában, a Schema ágat kifejtve állítsuk be a baseschema *(urn:schemas-microsoft-com:exch-data:baseschema)* propertyt a kívánt értékre. A fenti példával élve, a SchemaFolderC-re ez tehát a non_ipm_subtree/schema/, a SchemaFolderA-ra pedig a két, általunk létrehozott séma.

Ha saját, nem MAPI foldertreet használunk, a fent leírtak annyiban módosulnak, hogy a non_ipm_subtree/schema/ helyett a http://myserver/mynonMAPIFolderTree/##SchemaURI##/microsoft/exchangeV1/ helyen keresendő a WSS alapsémája.

Ha ezzel készen vagyunk, elkezdhetjük felvenni a saját sémánkhoz a saját jellemzőinket, majd a saját content classainkat. Mint azt fentebb már írtam, mindenképp célszerű ezeket névterekbe szervezni. A névterek legyenek beszédesek, és inkább hosszúak, mint szűkszavúak. Általában cégnév:alkalmazásnév:funkciócsoport felépítést ajánlott használni. Így egyrészt első ránézésre látszik, mire, milyen környezetben használjuk, másrészt teljes bizonyossággal elkerülhetők az ütközések, elnevezésbeli konfliktusok.
Ha megvan minden property és content class, akkor jöhet a következő lépés: az alkalmazásunkhoz hozzá kell rendelni a sémát. Előtte azonban nézzük meg az Exchange Explorer segítségével az alkalmazásunk mappájához tartozó Schema Propertyket *(Exchange Store Hierarchy nézet, jobb egérgombbal klikk a mappán, majd Schema Properties)*: azt láthatjuk, hogy csak beépített jellemzők szerepelnek a felsorolásban. Ez logikus is, hiszen ennek a mappának még semmit nem mondtunk a saját készítésű sémánkról, honnan is tudhatna róla? Viszont ha beállítjuk a mappa schema-collection-ref jellemzőjét *(urn:schemas-microsoft-com:exch-data:schema-collection-ref, Exchange Explorer, Detail View, Schema ág)* az elvárásainknak megfelelő séma mappára, akkor ez a lista máris kibővül a mi saját jellemzőinkkel. Sőt, a hierarchikus sémaszerkezetnek köszönhetően a hierarchiában lentebb lévő sémák jellemzői is megjelennek itt. *(Például ha egy ApplicationFolderre beállítjuk az előző példa SchemaFolderA sémáját, akkor az örökli a SchemaFolderB, a SchemaFolderC, sőt a WSS alapséma jellemzőit, content classait is.)*



♣ *A Web Storage System sémahierarchiája*

Természetesen egy séma több alkalmazáshoz is felhasználható, a hierarchiában elfoglalt helyétől függetlenül. Így akár kusza, bonyolult sémaszerkezetek is kialakíthatók a WSS-en belül.

Miért előnyös saját sémát definiálni?

Ez a Schema Properties lista látványos, ám kevésbé hasznos az alkalmazásunk szempontjából, hiszen ha létrehozunk valamit, annak - ideális esetben - kézzelfogható okai vannak.

Mi előnyünk származik tehát abból, ha használjuk a Web Storage System sémáit, és nem csak vaktában hozzuk létre a jellemzőket? Ha egy mappában létrehozunk új elemeket, azok content classa egy alapértelmezett értéket vesz fel a WSS alapsémájából. Ha viszont explicit kijelentjük, hogy az új elem content classa egy saját, általunk létrehozott érték legyen, néhány dolog megváltozik.

Ha az Exchange Explorerben követjük a változást, láthatjuk, hogy az elemhez rendelt jellemzők halmaza némileg megváltozott, azonban a mi saját jellemzőink nem kerültek fel a listára. De nem kell megijedni, ennek igen egyszerű és logikus magyarázata van. A WSS ugyanis csak azokat a jellemzőket tárolja, amelyeknek van értékük. Így egyrészt takarékosan bánik a hellyel, másrészt *(és ez a fontosabb)*, a hatékonysága is lényegesen jobb, mintha minden propertyt eltárolna, a hozzá tartozó üres értékkel együtt.

Természetesen mi magunk felvehetjük a szükséges jellemzőket akár kézzel az Exchange Explorerből, akár az alkalmazásunkból.

De mivel több ez, mint ha séma nélkül, egyszerűen csak „találomra" felvesszük a jellemzőket az elemekhez, és úgy dolgozunk velük? A különbség ott van, hogy ha egy alkalmazás mappája egyéni sémával rendelkezik, a sémához tartozó content classok, propertyk is hozzá rendelődnek. Tehát ha egy egyéni content classú elemre a programunkból kiadunk egy SELECT * utasítást, nem az eredeti jellemzőhalmazt kapjuk vissza *(mint saját séma használata nélkül)*, hanem az általunk, a sémában definiáltat. Ez azért jelentős könnyebbség, mint ha a SELECT után egyesével, mindent fel kellene sorolnunk. Ráadásul így a WSS mappánk határozza meg a találati listát, hiszen a séma változásával az eredményhalmaz is dinamikusan változik.

Másik fontos érv lehet a séma használata mellett a konzisztencia. Tegyük fel például, hogy egy elemhez hozzárendelünk egy string típusú ID-t, és azzal dolgozunk. A következő elem felvételekor azonban már integer típusú ID-t veszünk fel *(vagy vesz fel jóakaratú kollégánk)* ugyanazon a néven. Most akkor melyik is a valódi, melyiket is használjuk? Erre a problémára is megoldást kívál a séma használata, hiszen minden jellemzőnek előre definiált, jól meghatározott típusa van.

Ha a sémánkat jól dokumentáljuk, és a névtereket is következetesen használjuk, az is kiküszöbölhető, hogy ugyanazon célból több, különböző néven szereplő jellemzőt felvegyünk.

Mindezek a tulajdonságok elősegítik a csapatmunkát is. Ha többen dolgozunk ugyanazon a projekten, elkerülhetetlen, hogy az adatainkat közös megegyezés, közös elvek alapján használjuk, különben olyan káosz alakul ki az alkalmazásunk körül, amilyet elképzelni sem tudunk. Ha azonban kialakítjuk a propertyk, content classok tárházát, vagyis a sémát, minden fejlesztő egy helyen láthatja a már meglévő adatstruktúrát. Így nem jelenthet gondot az egyeztetés, az együttműködés.

Néhány érdekesség

Korábban szó volt arról, hogy a WSS mindent elemként tárol el, az elemeknek pedig minden esetben van content classuk, és minden esetben jellemzők egész sora van hozzájuk rendelve. Nincs ez másképp magukkal a propertykkel és content classokkal sem, ezt láthattuk, amikor az alapsémát nézegettük. Egy-egy property definíció content class-a urn:content-classes: propertydef, a content class-oké pedig urn:content-classes:contentclassdef. Nyilván lennie kell olyan jellemzőknek, amelyek lezárják ezt a „sort", hiszen ha minden property-definíciónak vannak propertyjei, azokat is definiálni kell valahol, és az ördögi kör bezárult. Nos, egyes definíciós elemek különleges jellemzőkkel rendelkeznek. Ezek névtere az urn:schemas-microsoft-com, nevük két összetevőből áll, ezeket '#' választja el egymástól. Például a jellemző nevét az urn:schemas-microsoft-com:xml-data#name határozza meg, a típusát pedig az urn:schemas-microsoft-com: datatypes#type. A property többi jellemzője az urn:schemas-microsoft-com:exch-data: névtérben található, és azt mutatja meg, hogy például felvehet-e több értéket *(ismultivalued)*, indexeljük-e a hatékonyabb kereséshez *(isindexed)*, stb.

Hasonló a helyzet a content classokkal is, ott is vannak ilyen és ehhez hasonló, magának a rendszernek szóló jellemzők.

A WSS séma tehát hétköznapi gondolkodásunkhoz hasonló eszmefuttatást igényel, nyilván sokkal formálisabb módon. Ha megértjük működését, már gyermeki egyszerűséggel építhetjük saját alkalmazásaink sémáját, és dolgozhatunk hatékonyabb, egyszerűbb módon, mint korábban.

Sok sikert hozzá!

Molnár Ágnes
agnes.molnar@t-systems.com

Webstatisztika SQL Serverrel?

Naplófájlok, regisztrációs listák elemzésére kész eszközök közül is választhatunk, különféle scripteket *(VBScript, JScript)* is írhatunk, de ezek egyike sem ér fel az SQL, mint halmazorientált nyelv könnyedségével. Ha ellapátoljuk utunkból az akadályokat, huszonvalahány soros VBScriptek helyett egysoros SQL utasításokkal elérhetjük ugyanazt az eredményt...

Manapság minden második weblap adatokat gyűjt. Rendeléseket, véleményeket, e-mail címeket stb. Az adatok tárolási helye néha egyszerű textfájl, de az esetek elsöprő többségében valamilyen adatbázis. Kézenfekvőnek tűnik az SQL Server beépített lehetőségeinek használata. Hogyan tudnánk az SQL Server saját eszközeivel feltárni az összefüggéseket? Ebben a cikkben a Transact SQL változatos lehetőségeiben tobzódunk.

A kihívás

Képzeljük el, hogy egyik weblapunk e-mail címeket gyűjt az arra járó látogatóktól, amelyek egy SQL táblába kerülnek. Ha az lenne a feladatunk, hogy mondjuk meg, hány regisztráció érkezett Japánból, vagy akár a @kulupintyo.hu címről, rögtön szembetalálnánk magunkat azzal az alapproblémával, amit az amerikai gondolkodásmód okoz. Nevezetesen: minden hierarcikus adathalmazt jobbról balra fejtünk ki - miközben balról jobbra írunk!

A kelemen@falatrax.hu *(ez Kőműves Kelemen e-mail címe)* címet megvizsgálva belátható a fenti állítás. A tartománynevek hierarciájában a „hu" feljebb van, mint a „falatrax". Magyar ember a DNS-fát valószínűleg így fejtené ki: hu.falatrax.www. Ebben az esetben ugyanis az olvasási sorrend megfelel a hierarchia kibontásának is. De az amcsi nem ilyen. Amit lehet, fordítva ír, hogy aztán a számítógépes feldolgozásnál lehetőleg gond legyen. Gondoljunk csak a dátumokra *(13.11.2003.)* vagy a lakcímekre *(34234. 546. ez utóbbi az utca „neve")*.

Rövidre zárva: hogyan keressük ki a .jp végződésű *(Japán)* e-mail címeket a táblából? Hogyan választható le a címből a TLD *(Top Level Domain Name)*? A LIKE operátorral? Valahogy így?

```
SELECT email from tabla
WHERE email LIKE '%.jp'
```

Ez a lekérdezés kétségkívül alkalmas a Japán címek leválogatására, de:

- Nem általánosítható. Nem lehet vele az előforduló TLD-ket kilistázni. Bele van varrva a keresési feltétel, esetünkben Japán. Ez egy Japán-kereső egyedi megoldás. *(Könnyen átírható akár Madagaszkár-keresővé is, de ember legyen a talpán, aki összehozza azt a reguláris kifejezést, ami általánosá teszi, és minden TLD-t lefed!)*
- A LIKE operátor mindenütt használható, ahol az operátorok - tehát csoportosításban, aggregációban NEM! Ezzel a módszerrel nem leszünk képesek olyan listák előállítására, ahol a kimenet egyik oszlopában a különböző TLD-k sorakoznak, a másikban pedig előfordulási darabszámuk.
- Nagy adatmennyiségek esetén, illetve JOIN feltétel részeként - hihetetlenül lassú. *(A későbbiekben az e-mail címek TLD-je alapján össze kívánjuk kötni a sorokat egy másik táblával!)* Tessék mondani, milyen indexet tudunk használni a fenti lekérdezés elvégzésekor? Ha van egy fantasztikus egyedi indexünk az email mezőn, gyorsabbá válik a fenti lekérdezés? Nem? Vagy mégis?

Milyen a jó keresési feltétel?

Frissítsük fel indexekről szóló, megfakult elméleti tudásunkat! Vegyünk *(kézbe)* egy telefonkönyvet! Ez a vaskos névjegyzék [vezeték, keresztnév] összetett index szerint van rendezve, azaz a nevek először családnév szerint, azonos családnéven belül keresztnév szerint állnak sorban. A keresési sorrend megegyezik a nyomtatási *(fizikai)* sorrenddel: a telefonkönyvön clustered index van. Hogyan keresünk ebben „Fóti" nevű egyéneket? Egyszerűen felütjük az „F" környékén, néhányat lapozunk előre-hátra, míg pontosan meg nem találjuk a megfelelő lapot, azon belül pedig addig húzzuk lefelé az ujjunkat, amíg meg nem találjuk a célszemélyt. Ilyen egyszerű! És hogyan keresünk benne „Marcell" keresztnevű embereket? Ne feledjük, van egy [vezeték, keresztnév] indexünk! Mennyiben segíti ez a keresést? Semennyiben! Ha tud valaki a szorgos végiglapozásnál *(Table Scan)* jobb módszert arra, hogy kikeresse egy keresztnév összes előfordulását, szóljon nekem! Milyen sorrendben találjuk a [vezeték, keresztnév] összetett indexben magukat a keresztneveket? Úgynevezett kaotikus sorrendben...

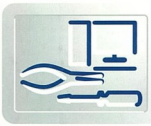
A feladat gyors elvégzéséhez egy JÓ index kellene! Ha valaki hozzájut a telefonkönyv CD-s változatához, a keresztnév-keresés nem okoz gondot, mert a CD-n már van önálló [keresztnév] index is.

A telefonkönyves gondolatmenet segítségével látható be, hogy az email mezőn lévő index nem alkalmas a LIKE '%.jp' kérdés megválaszolására:

- Ha egy *(szöveges)* mezőn index van, a mező értékei ABC sorrendben érhetők el *(triviális)*. *(A charset - sort oder rémálmot most hagyjuk figyelmen kívül!)*
- Ha egy *(szöveges)* mezőn index van, a mezők legelső karaktere ABC rendben követi egymást, így a LIKE 'a%' lekérdezés gyorsítható a segítségével. Minden „balról zárt" keresés gyorsítható indexszel.
- Ha egy *(szöveges)* mezőn index van, a mező belső *(sokadik)* karakterei kaotikus sorrenben állnak az indexben! Balról nem zárt keresések gyorsítására az index nem alkalmas! Így a LIKE '%.jp' kérdés nem tudja kihasználni az email mezőn létrehozott indexet!

Indexeljünk jobbról balra?

Ezek a szerencsétlen e-mail címek jelenlegi formájukban min-



den indexelési kísérletnek ellenállnak. Van ugyan az SQL Serverben egy csodálatos függvény stringek megfordítására *(a REVERSE)*, de ez sajnos karakterenként fordít *(kelemen@falatrax.hu = uh.xartalaf@nemelek)*. További példák:

```
SELECT REVERSE('Indul a g r g aludni')
SELECT REVERSE('RÆm nØmet nem lel, elmentem
 Øn mÆr')
```

Ez a módszer sem működik tehát. Valahogy rá kellene venni az SQL Servert, hogy az utolsó két karakterre indexeljen! Ilyet utoljára a Clipper tudott, mert azzal tetszőleges függvény alapján lehetett indexelni. Az SQL Server azonban csak mezőre tud indexet építeni...

Számított mezők

A megoldás kulcsa az SQL Server számított mezőinek *(computed columns)* felhasználásában rejlik. Az igaz, hogy az SQL Server csak mezőket tud indexelni, ám egy-egy mezőbe természetesen bármi kerülhet. Ha ráadásul kihasználjuk az SQL Server azon képességét, hogy röptében képes mezőértékeket kiszámítani, nem kell sem triggerrel, sem más módszerrel vacakolnunk. Magába a tábladefinícióba belekombinálhatjuk a megfelelő számítási formulát, és van is új mezőnk - meg nincs is. Jól használható a számított mező olyan esetekben, amikor egyik mező valami fix kalkulus révén egy másikból származtatható *(pl.: ár -> áfás ár)*, de nem kívánjuk az így előállt új értéket sem tárolni, sem módosítani.

Figyelem! A számított mező értéke nem tárolódik, hanem minden egyes lekérdezésnél újra és újra kiszámításra kerül!

A megfelelő számítás kifundálásához nem árt, ha megismerkedünk végre annak a táblának a szerkezetével, amelybe a webről gyűjtött adatok potyognak:

cimek

Column Name	Data Type	Length	Allow Nulls
webcim	varchar	255	✓
email	varchar	255	✓

♣ ***A cimek tábla két mezője***

Nem bonyolította túl a fejlesztő, annyi szent, de legalább könnyedén áttekinthető. Távlati célként fogalmazzuk meg, hogy a cimek táblát összekapcsoljuk egy másik táblával, mégpedig a jelentkező e-mail címéből kivont TLD alapján. Ez utóbbi tábla is észveszejtően bonyolult - ezt szokták kiérdemelni a weblapok mögé odahányt tablák:

tld

Column Name	Data Type	Length	Allow Nulls
kod	varchar	5	✓
neve	varchar	25	✓

♣ ***A TLD tábla soronként egy Top Level Domainnevet és egy megnevezést tartalmaz***

A számított mező értékének előállítására egyetlen determinisztikus képletet *(függvényt)* kell megadnunk. Ehhez szükségünk lesz néhány stringfüggvényre. Az eljárás menete röviden: megkeressük az email címben az utolsó pontot, és innentől kivágjuk a hátralévő összes karaktert. Mondani könnyű, de csinálni már nem annyira, ugyanis csak „balos" stringkeresőfüggvényt *(CHARINDEX)* tartalmaz a TSQL nyelv, s annak nem lehet megmondani, hogy az utolsó előfordulásra vagyunk kíváncsiak. Szerencsére itt van a jó öreg REVERSE, s ezzel már előrekerül az a fránya pont:

```
SELECT REVERSE('kelemen@falatrax.hu')
------------------

uh.xartalaf@nemelek

(1 row(s) affected)
```

Ha erre ráeresztjük a CHARINDEX függvényt, mely egy bizonyos mintát *(esetünkben egy pontot)* keres egy stringben, a következő eredményt kapjuk:

```
SELECT CHARINDEX('.',
 REVERSE('kelemen@falatrax.hu'))
-----------

3

(1 row(s) affected)
```

Közelítünk, közelítünk! Ez a fantasztikus képlet visszaadja, hogy hátulról hanyadik karakter a pont *(ha van)*. Nincs más hátra, mint az így kiszámított utolsó x darab karakter kihasítása az eredeti stringből a SUBSTRING segítségével, amihez még fel kell használnunk a string hosszát visszaadó függvényt *(LEN)* is. Ezzel eljutunk a következő tökéletesen áttekinthető egyszerű függvényhez:

```
SELECT SUBSTRING('kelemen@falatrax.hu', LEN('kele-
men@falatrax.hu')-
CHARINDEX('.', REVERSE('kelemen@falatrax.hu'))+2,
CHARINDEX('.', REVERSE('kelemen@falatrax.hu'))-1)
-------------------

hu

(1 row(s) affected)
```

Jól jegyezzük meg a fenti képletet, mert a cikk hátralevő részében egyre bonyolultabb pozitúrákban rendszeresen visszatér! Fel kell majd ismerni!

(A plusz kettő azért kell, mert egyfelől a SUBSTRING nem nullabázisú, másfelől a TLD előtti pontra sincs szükségünk. A mínusz 1 értelemszerűen a string túlcímzését előzi meg - bár ez ellen a SUBSTRING amúgy is védett. Nem érdemes erre apellálnunk, mert az, hogy a jelenlegi verzió védett, nem jelent semmilyen jövőbeni ígéretet!)

Emlékeztetőül: a fenti borzadály csak annyiban jobb a LIKE operátornál, hogy:

1. általános módja a TLD képzésének *(nincs belevarrva semmilyen országstring)*
2. függvény, és nem operátor, így számított mező alapját képezheti

A fenti képletet már alkalmazhatjuk is a cimek táblán:

```
SELECT SUBSTRING(webcim, LEN(webcim)-
CHARINDEX('.', REVERSE(webcim))+2,
CHARINDEX('.', REVERSE(webcim))-1)
FROM cimek
```

Ad-hoc lekérdezés a tld-k kilistázására (1)

Hosszú és áttekinthetetlen, de a mienk! Ha a képlet segítségével számított mezőt képezünk, ez a csinos, négysoros lekérdezés így nézne ki:

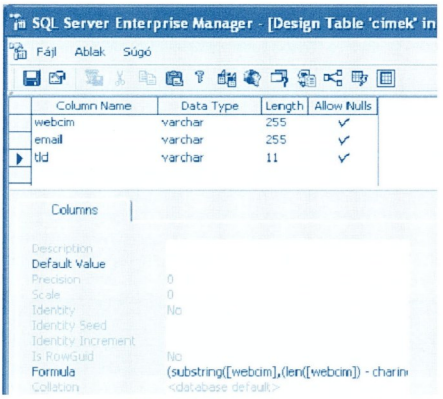
```
SELECT tld FROM cimek
```

Számított mezőn alapuló változat a tld-k kilistázására (2)

Áttekinthetőbb, nemde? Bárcsak lenne egy ilyen mezőnk! Ennyit kell tennünk:

```
ALTER TABLE cimek ADD tld AS
SUBSTRING(webcim, LEN(webcim)-
CHARINDEX('.', REVERSE(webcim))+2,
CHARINDEX('.', REVERSE(webcim))-1)
```

Vagyis a fent bemutatott függvényt egy tld nevű, számított mező alapjává tesszük. Aki az egérben bízik jobban, dizájnolja a táblába a képletet az Enterprise Manager segítségével!



♣ Számított mező készítése Enterprise Managerrel

Teljesítménykérdések

Vajon milyen teljesítményt nyújt a számított mezőt használó *(2)*, sokkal rövidebb lekérdezés? Talán nem meglepő, hogy szinte hajszálpontosan ugyanannyi ideig fut, mintha a függvénykomplexumot közvetlenül a lekérdezésbe írtuk volna *(1)*. Ennek az az oka, hogy a számított mező alaphelyzetben minden futáskor újra és újra kiszámítódik, nem mintha más lenne az értéke, de a számítás eredményét az SQL Server nem tárolja.

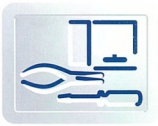
Készítetttem egy tízezer soros példatáblát, ahol mind a számított mezős *(2)*, mind az ad-hoc *(1)* lekérdezés 120 ms alatt futott le.

Most tegyünk indexet vadonatúj, tld nevű számított mezőnkre!

```
CREATE INDEX kompu ON cimek(tld)
```

Ezután a számított mezőre kiadott lekérdezés *(2)* sebessége négyszeresére *(31 ms)* nőtt!

Ebben az az izgalmas, hogy maga a lekérdezés nem igazán index-barát, mivel - WHERE feltétel híján - a tábla összes sorát visszaadja a kimenetre. Korábbi teljesítményfeltáró cikkeink nyomán azt mondhatná Kedves Olvasóm, hogy egy ilyen lekérdezésen semmit sem segít az index. Ez általában igaz is. Számított mezőnél azonban valami más változást is hoz az index: a mező értéke MATERIALIZÁLÓDIK, kiszámításra és raktározásra kerül!



Fontos!
A számított mezők értéke alapesetben nem tárolódik, de ha indexet teszünk rá, kényszerűen tárolásra kerül, hisz indexkulcsot kell képezni az adatokon.

A számított mező értékének kiszámítását így egyszer s mindenkorra letudta az SQL Server, s ha kérdezzük, már az indexből olvassa vissza a konzerv-adatokat.

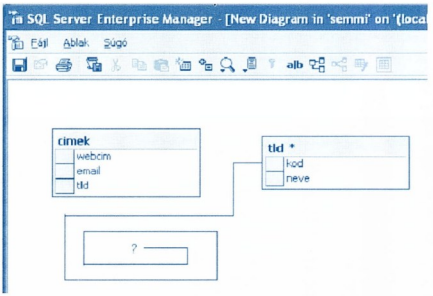
Illeszések *(JOIN)*

Már csak egy dolog van hátra: illesztés végzése a számított mező alapján. Ne zavarjon minket, hogy ez a tld mező semmilyen kulcsjellemzővel *(PRIMARY, FOREIGN)* nem bír. Ettől még JOIN alapját képezheti. Sőt! Akármilyen kifejezés JOIN alapja lehet, tehát a számított mező ad-hoc elődje is, mint azt az alábbi SELECT tanúsítja:

```
SELECT SUBSTRING(webcim, LEN(webcim)-
CHARINDEX('.', REVERSE(webcim))+2, 11), neve
  FROM cimek INNER JOIN tld ON
    SUBSTRING(webcim, LEN(webcim)-
    CHARINDEX('.', REVERSE(webcim))+2,
    CHARINDEX('.', REVERSE(webcim))-1) = tld.kod
```

Illesztés a tld táblához ad-hoc képlettel (3)

Ezt a joint talán így ábrázolná a grafikus Designer *(az ábra hamisítvány)*:



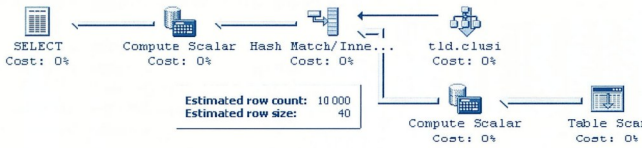
♣ Ne keverjük össze a JOINT a referenciális integritással! A kettő nem ugyanaz!

Ez azért van, mert összekevertük a szezont a fazonnal. Illeszteni *(az adattípusok egyezésének biztosításával)* bármit bármivel lehet. Az illesztés egy-egy ***lekérdezésre*** vonatkozik. Ennek semmi köze a táblák közt fennálló referenciális integritáshoz, melynek az adatok ***felvitelekor*** van lényeges szerepe! Jó-e az alábbi JOIN, amit a Northwind adatbázis két tábláján alkottunk? *(Figyeljük meg a JOIN feltételt, ON 3 = 4!)*

```
SELECT * FROM employees INNER JOIN orders ON 3 = 4
```

Jó. Értelmetlen? Nem biztos. Aki nulla sort vár egy illesztett lekérdezéstől, ezzel a feltétellel biztosra mehet. De ennek semmi köze a táblák közt fennálló referenciális integritáshoz!

Visszatérve az illesztésekhez: az ad-hoc képletes és a számított mezőre épített JOIN futása között egyetlen lényeges különbség fedezhető fel: az index használata! Most két SQL végrehajtási tervet mutatok be. Az első a *(3)* számú, ad-hoc képletes JOIN végrehajtásának lépéseit mutatja, míg a második a számított mezőre *(és ezáltal annak indexére)* támaszkodó futás során keletkezett.



♣ ***Illesztés bonyolult ad-hoc képlet (3) mentén***

Vegyük észre, hogy szegény SQL Server mennyit fontoskodik, mire elérnek az adatok a felhasználóhoz *(az ábrát amerikai módra, jobbról balra kell értelmezni)*.

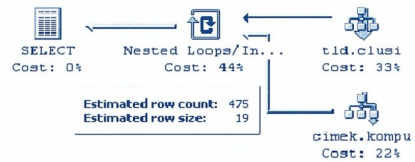
- Először is találunk egy Table Scan műveletet *(az egész cimek tábla végigolvasása sorról sorra)*
- Ezután következik a képlet kiszámítása minden sorra *(Compute Scalar)* az illesztés elvégzésének érdekében
- Ezt követi egy Hash Join illesztés, melyről tudjuk, hogy akkor veszi elő az SQL Server, ha a lekérdezés támogatására nincs jó indexünk. Ha Hash Joint látunk, gyanakodjunk indexhiányra!
- A join után mégegyszer lefut a képlet, mert szegény SQL Serverünk nem veszi észre, hogy az outputra ugyanaz az eredmény kerül, mint ami a join alapját képezte

Hajszálpontosan ugyanazt az adathalmazt juttatja el a kimenetre az alábbi lekérdezés, de ez már támaszkodik a számított mezőre és annak indexére:

```
select tld, neve
from cimek inner join tld on tld = tld.kod
```

Illesztés a tld táblához számított mezővel (4)

Ennek végrehajtási terve lényegesen egyszerűbb. Fogjuk az indexet mindkét táblán, és felhasználjuk az illesztéshez, mellesleg a számított mező indexkulcsai lesznek a kimeneti értékek: kettőt egy csapásra!



♣ ***Ugyanaz az illesztés indexelt, számított mezőre***

A figyelmes szemlélő esetleg megbotránkozik, hogy ez a tökkelütött SQL Server miért Nested Loop stratégiával áll neki a feladatnak, miért nem Merge Joint használ, noha mindkét táblához van index. Ha azonban ráerőszakoljuk erre a lekérdezésre a saját okosságunkat, és beleszögelünk egy MERGE módosító operátort, a futási teljesítmény drasztikusan *(harmadára)* romlik. Sőt! Munkatáblát állít fel magának az SQL Server, aminél teljesítményrombolóbb megoldást nehezen tudunk elképzelni!

A két lekérdezési terv *(Hash és Nested)* között még egy lényeges, szembetűnő különbség fedezhető fel: az ikonokat összekötő nyilak vastagsága *(Estimated row count)*. A bonyolultabb Hash Join mind a tízezer sort kiolvassa a cimek táblából, és gyakorlatilag a folyamat során végig tízezer sort tart a memóriában. Íme a tízezer buzogányt levegőben tartó zsonglőr!

Ezzel szemben a Nested Loop egyszerre, egy időben 4-500 sort kezel, mert egy tld-hez átlagosan ennyi emailcím tartozik *(ez az indexstatisztikák eloszlásfüggvénye alapján is megállapítható lenne)*.

Hány Japánunk van?

Végül ne feledkezzünk meg az eredeti kérdésről sem: hány Japán érdeklődőnk van? Ehhez - minden különösebb magyarázat nélkül - a GROUP BY záradékot és a COUNT számlálófüggvényt fogjuk használni. Erre két, azonos eredményt adó lekérdezést is írhatunk *(az eltérést kiemeltem)*:

```
SELECT tld, COUNT(tld)
  FROM CIMEK
    WHERE tld='jp'
      GROUP BY tld
```

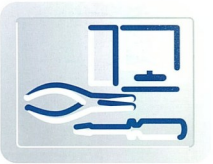
és

```
SSELECT tld, COUNT(tld)
FROM CIMEK
GROUP BY tld
HAVING tld='jp'
```

Ha egyforma eredményt adnak, mégis, mi a különbség? A megfejtéseket levélben várom az alábbi címen:

Fóti Marcell
marcellf@netacademia.net
MCDBA, MZ/X

Modellezés, tervezés és analízis (1. rész)



Bevezetés

Kezdő programozó koromra visszatekintve azt látom, hogy „fejlesztési" szokásaim igencsak eltértek a mostaniaktól. Tervezés nélkül, többnyire in medias res belevágtam a dolgokba, „úgyis megy ez nekem". És mivel ezek többnyire kisebb programocskák voltak - ment is.

Aztán egyre növekedtek az önmagammal szemben támasztott igényeim, egyre magasabb röptű elhatározásokra jutottam, egyre merészebb programokat kezdtem írni - természetesen továbbra is előzetes tervezés nélkül. Egyszer azonban elérkezett a varázslatos pillanat: az első memória-túlcsordulás, a reménytelen, éjszakába nyúló debuggolások, és a találgatások *(vajon hol lehet a hiba?)* időszaka. Rá kellett döbbenem: ez így nem mehet tovább.

Természetesen a „nagyok" módszereit ekkor még hírből sem ismertem, így saját fejem szerint kezdtem első „terveimet" elkészíteni. Az eredmény: sokkal hatékonyabb programocskák. A következő lépés az volt, amikor már másokkal együtt végeztem a munkámat, és kezdtem észrevenni *(és szóvá tenni)*, ha egy-egy ötlet, terv nem jó, vagy esetleg kifejezetten rossz volt: érthetetlen, logikátlan, ellentmondásos, nem egyértelmű stb. *(Persze ekkor meg az a vád ért, hogy túl kötekedő vagyok...)* Szerencsére azonban volt szerencsém megismerni több formális módszert is, és bekövetkezett az elkerülhetetlen: én magam is elkezdtem szoftvereket tervezni. Így ma már a szoftverfejlesztés teljes menetére rálátásom van. Nem mondom, hogy minden egyes lépést tekintve „profi" vagyok, de szert tettem számos olyan tapasztalatra, amelyet szeretnék most közzétenni.

A szoftvermérnökség (software engineering)

> „Engineering:
> Creating cost-effective solutions...
> ...to practical problems...
> ...by applying scientific knowledge...
> ...building things...
> ...in the service of mankind."
> - M. Shaw -

A fejlesztés során tehát ezen mérnöki eszközöket használjuk fel, innovatív gondolataink és rutinná vált módszereink ötvözésével. A feladat nem más, mint a követelményeknek és a korlátozásoknak *(anyagi, erőforrásbeli, emberi, stb.)* megfelelő rendszer kidolgozása, megalkotása, üzembe helyezése.

♣ ***Az előttünk álló feladat***

Ez az út azonban meglehetősen hosszadalmas. A termék fejlesztése mellett nagyon fontos a megfelelő dokumentáció *folyamatos* készítése is, hiszen egyébként 1-2 hét elteltével már magunk sem tudjuk mit csináltunk, hogyan, és főleg miért azt és miért úgy.

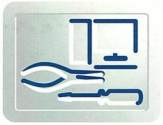
Alapvetően két nagy csoportról, felhasználói és fejlesztői dokumentációról beszélhetünk. Az előbbi a rendszert használó és megrendelő, a fejlesztési folyamattól távol álló *(és ahhoz nem értő)* emberekhez közérthető nyelvezeten szól, kerülve a formalizmusokat és bonyolult szakkifejezéseket. Az utóbbi éppen ellenkezőleg, belső dokumentáció, a fejlesztőknek szól, a szakma nyelvén, lehetőleg minél egyértelműbb, absztraktabb formális módszereket felhasználva.

Az analízis

A kész termékhez vezető úton először is meg kell határoznunk, mi is a pontos feladat. A megrendelővel történő hosszas egyeztetések során először a Rendszer definíció születik. Ennek tartalmaznia kell a felhasználó igényeit, a megvalósítandó funkciók halmazát, céljait és indokait. Itt kell leírni az előre látható korlátozásokat, amelyeket például a felhasználó pénztárcája, pillanatnyi felindultsága vagy lelkesedése támaszt. Definiálni kell azokat a feltételeket, amelyeknek meg kell felelnie a készítendő rendszernek ahhoz, hogy késznek mondhassuk és átadhassuk. Egyeztetni kell a szakkifejezések használatát is a félreértések elkerülése érdekében.

A következő lépés a *Projekt terv* elkészítése. Ez már sokkal közelebb áll a fejlesztésben résztvevőkhöz is, de közel marad az egységsugarú megrendelőhöz is. Tartalmazza a becsült fejlesztési ütemtervet, a személyi és erőforrás követelményt, a becsült költségeket. Ehhez figyelembe kell venni különböző szakértői véleményeket, korábbi tapasztalatokat és egyéb módszereket.

A Projekt tervben határozzuk meg a felhasznált eszközöket, módszereket, programozási nyelveket, a tesztelési követelményeket és az igényelt dokumentációt. Fontos rögzíteni azt, hogy a megrendelő milyen határidővel, esetleg



milyen részletekben, mennyit fizet az elvégzett munkáért, és hogy ezért pontosan milyen módon kapja meg a rendszert *("dobozos" formában, beüzemelve, esetleg karbantartást, módosítást is vállalunk a későbbiekben stb.).*

Követelmények konkretizálása

A megrendelővel együtt készített megvalósíthatósági vizsgálatok után rögzítjük a rendszer *Követelmény definícióját*, amely már konkrét adatszerkezeteket, adatfolyamokat, akciódiagramokat tartalmaz. Ez már egy állapot-orientált leírás, tehát a rendszer állapotait, az azok közötti átmeneteket helyezi középpontba. Ehhez rendkívül jól használhatók a különböző reguláris kifejezések, és például a Petri hálók *(ezekről majd egy későbbi cikkemben szólok).*

A *Verifikációs terv (verifikáció: helyesség ellenőrzése, bizonyítása)* a teljesítendő követelmények, a tervezés, a forráskód, és a dokumentációk helyességének feltételeit, a tervezés befejezésének feltételét foglalja magában.

Itt kezdődhet a *Felhasználói kézikönyv* megírása is, amely ebben a fázisban általános információkra, áttekintésekre, elemzésekre, működési módokra stb. terjedhet ki.

Tervezés

Ha a követelményeket is világosan definiáltuk, elkezdődhet a tervezés, amely már nem a felhasználónak, hanem a fejlesztésben részt vevő belső munkatársaknak szól. A specifikált követelményeket felhasználva ebben a fázisban a program struktúráját és annak elemeit határozzuk meg. Cél az absztrakciós szint csökkentése, a dolgok konkrétabbá tétele, a formalizálás. Itt születik meg az *Architektúra terv*. Ebben döntünk az adatszerkezetek és adatbázisok megvalósításáról, az alapobjektumokról, konkrét algoritmusokról, az esetleges különleges programozási módszerekről és kivételes állapotokról. Ezen absztrakció révén figyelmen kívül hagyjuk a probléma szempontjából jelentéktelen részleteket, és csak a lényegre koncentrálunk.

Ebben a fázisban születik meg a részletes *Felhasználói kézikönyv*, és a végleges *Verifikációs terv* is. A Részletes terv alapján elkezdődhet a konkrét programozás...

Implementálás

Az implementáció a specifikáció direkt végrehajtása, az algoritmusok formalizálása valamely programnyelven vagy -nyelveken. Általában több programozó dolgozik egy projekten, így nagyon fontos az előzetes tervek egyértelműsége, és a probléma jól elkülöníthető részekre bontása. Optimalizálni kell a munkában részt vevő emberek adottságaira, lehetőségeire és korlátaira is.

A tervezés során vegyük figyelembe az informatikában jól ismert *40-20-40 szabályt*: ez a százalékos aránya a tervezéssel, kódolással és teszteléssel eltöltött időnek egy igazán jól kidolgozott, alapos rendszer esetében. Az emberi erőforrásokat azért kell tudni jól meghatározni, mert egy későbben lévő projekt új ember beállításával többnyire további késedelmeket szenved *(a betanulás, átállás nemcsak az új embert, de a már „beavatottakat" is igénybe veszi).*

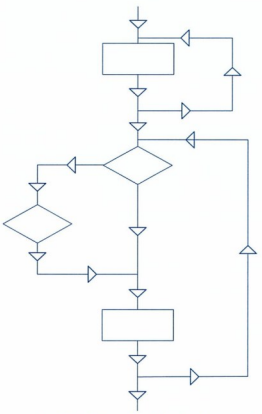
Tesztelés

A már *(részben)* kész szoftvert mindenképp tesztelni kell, hiszen tökéletes programozó még nem született erre a világra. *(Valamikor május elején jött az egyik listára: „Örökös versenyfutás folyik a programozók és az Úristen között: a programozók igyekeznek minél nagyobb, jobb és idióta-biztosabb programokat létrehozni, az Úristen pedig igyekszik minél nagyobb és jobb idiótákat gyártani. Egyelőre az Úristen áll nyerésre"...)*

Első körben már maguk a fejlesztők is végeznek valamiféle tesztelést, hiszen a programkód begépelése után *(többnyire)* lefordítják és futtatják azt. Mindenképp szükséges azonban az olyan célirányos, tudatosan hibákat kereső tesztelők munkája is, akik a kódot nem ismerik, és a tudatlan felhasználót képviselik, aki mindenre képes. A tesztelők lehetnek házon belüli, de külső, vállalkozó kedvű felhasználók is. Cél a program hibáinak felismerése, lokalizálása, és természetesen javítása.

Természetesen a tesztelés is előre tervezett módszerekkel, szisztematikusan történik, akár erre alkalmas tesztprogramok segítségével. A *kimerítő tesztelés* minden lehetséges bemenő adattal kipróbálja a program működését, teljes képet egyedül így kaphatunk. Persze a gyakorlatban ez kivitelezhetetlen, ezért többnyire a specifikáció alapján felállított ekvivalencia-osztályok alapján tesztelünk: egy osztályt alkotnak azok a bemenő adatok, amelyek a tesztelendő egység szempontjából hasonló tulajdonságúak. Ezek a csoportok a jó tervek és dokumentációk alapján könnyen meghatározhatók.

A *strukturális tesztet* a program vezérlési gráfja segítségével végezzük. Az alábbi folyamatábrán például láthatunk feltételes elágazást és ciklusokat is. A feltételeknél nem elég csak az egyik ágat megvizsgálni, a ciklusok magján nem elég egyszer végigmenni. A program általában több szekvencia mentén is végrehajtható. Végtelen sokszor nyilván nem próbálkozhatunk, a tesztelési terv feladata meghatározni, mi az a határ, ahol már elfogadhatónak tartjuk a rendszer viselkedését. *(Például egy-egy kritikus rész 10.000-szer történő sikeres végrehajtása esetén, vagy ha 1000 felhasználó egyidejű bejelentkezése sem lassítja számottevően a programot, stb.)*



♣ Hányféleképp hajtható végre a fenti folyamatábra programja?

A tesztelőktől többnyire egy-egy hibajelenség leírását kapjuk, a hiba okának felderítése és kijavítása a programozók feladata. Nagyon fontos a korrekt információáramlás, a hibák pontos dokumentálása, hiszen hatékony javítást csak így várhatunk el. A javítás után újra tesztelni kell az adott egységet, nehogy újabb hibákat vigyünk bele a rendszerbe.

Integráció, üzembe helyezés, karbantartás

A megrendelővel kötött előzetes megállapodás értelmében szükség lehet az elkészült rendszer üzembe helyezésére és későbbi karbantartására. Fontos, hogy az ezzel kapcsolatos feladatokat egyértelműen definiáljunk, hiszen egy-egy félreértés nagymértékben ronthatja megbízónkkal kiépített kapcsolatunkat. Pontosan le kell írni, milyen lépésekből áll az üzembe helyezés *(hardver és szoftver elemek installálása, a rendszer felhasználóinak oktatása, stb.)*. A karbantartási kötelezettségek egy részét az alaprendszer költségei fedezik, szóba jöhetnek azonban olyan módosítások, újítások, bővítések amelyek újabb megegyezés tárgyát képezik. Ezekről is világos megállapodást kell kötnünk.

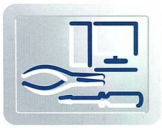
Követelmények a kész termékkel kapcsolatban

Felmerülhet azonban a kérdés: mikor nevezhetünk egy terméket készneknek? A minőségi szoftver több kritériumnak kell, hogy megfeleljen, ezek közül néhányat említenék:

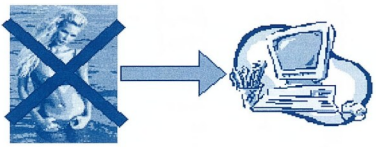
- **Helyesség:** A program működése megfelelő-e? Ez egyrészt helyes specifikáció, másrészt a specifikációnak megfelelő megvalósítás kérdése.
- **Robosztusság *(hibatűrés)*:** Egy rendszer hibatűrő, ha abnormális körülmények között sem fagy le, nem abortál, hanem „normálisan" lekezeli az előállt helyzetet, és erről tájékoztatja a felhasználót *(esetleg a háttérben az adminisztrátort vagy más különleges jogokkal rendelkező személyt is)*.
- **Hatékonyság:** A szoftver milyen mértékben használja ki a rendelkezésére álló hardver és szoftver erőforrásokat, mennyire optimális a háttértár, a memória és az idő szempontjából? Ezek a szempontok sokszor egymásnak ellentmondóak is lehetnek, ekkor meg kell találni a rendszer szempontjából legszigorúbb megkötéseket, és aszerint kell eljárni. Például egy adatbázis esetében a redundáns adatok növelik az elfoglalt tárterületet, azonban csökkenthetik például a kereséshez szükséges időt.
- **Karbantarthatóság:** Egy szoftver lényeges tulajdonsága, hogy később mennyire javítható, módosítható, illetve bővíthető új elemekkel. A jó program kódja könnyen áttekinthető, világos, egyértelmű szerkezettel rendelkezik.
- **Hordozhatóság:** Mennyire vihető át a termék más hardver- és/vagy szoftverkörnyezetbe?
- **Felhasználóbarátság:** A szoftver felhasználói felülete, megjelenése kellemes, használata kényelmes, logikus, egyszerű, könnyen megérthető és elsajátítható.
- **Integritás:** A különböző rendszerhibák milyen mértékben okoznak helyreállíthatatlan hibát a szoftver egyes részeiben, például az adatállományokban, a működéshez szükséges rendszerfájlokban, stb.

Természetesen a sor itt nem ér véget, végeláthatatlanul lehetne bővíteni a listát. Én ezt most nem teszem, mindenki saját ízlésére bízom, mit ad még hozzá, esetleg mit vesz el belőle *(ez utóbbi azonban nagy körültekintést igényel, hiszen bármely szempont elhagyása a minőség rovására mehet)*.

Modellek



Itt most nem Claudia Schifferről és társairól szeretnék írni *(bár alighanem ők is sokak fantáziáját lekötnék)*, hanem a különböző informatikai rendszerek modelljeiről.



♣ Informatikai rendszerek modelljeiről van szó...

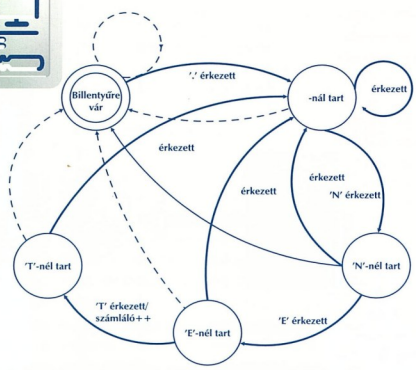
Először tisztáznunk kell, miért van szükség rendszerek modellezésére, hiszen a legtöbb szoftver már maga is a valóság egy részének modellje. Van egyáltalán értelme a modell modellezéséről beszélni? Természetesen van!

A *modell* „azon céllal létrehozott konstrukció, hogy rajta a valóság bizonyos jelenségeit jobban tanulmányozhassuk" illetve „a vizsgált rendszer vagy folyamat belső összefüggéseit, legjellemzőbb sajátságait, rendszerint matematikai egzaktsággal képletekbe sűrítő formula" *(Idegen szavak és kifejezések szótára, 1986.)*. Ha egy informatikai rendszer működését kívánjuk tehát modellezni, annak leglényegesebb tulajdonságai megtalálásával, kiragadásával, összefüggései feltárásával építhetjük fel azt. Célja kettős lehet: egyrészt egy már létező, kész rendszer működésének megértése *(analízis)*, másrészt egy készülő rendszer majdani működésének leírása.

Az informatikában használatos modelljeink többnyire valamilyen grafikus nyelven írják le a valóságot, persze vannak matematikai formalizmusra építő módszerek is *(például a különböző temporális logikák, deklaratív nyelvek stb.)*.

Nézzünk egy példát a könnyebb érthetőség kedvéért: tekintsünk egy egyszerű szövegszerkesztő programot, amelynek feladata a szövegben előforduló *„.NET"* betűsorozat számlálása. A szöveget folyamatosan visszük be a billentyűzetről, és a program a keresendő szöveg minden egyes előfordulásakor egy számláló értékét növeli eggyel. Az egyszerűség kedvéért tegyük fel, hogy a billentyűzetről csak betűket és pontot *('.')* viszünk be, a számláló értéke kezdetben nulla, és a szövegben visszafelé törölni nem lehet. Természetesen egy valós rendszerrel szemben ezek irreális korlátozások, hiszen hogyan is kényszeríthetnénk a felhasználót arra, hogy csakis és kizárólag betűket gépeljen, ne töröljön semmit, amit egyszer már begépelt, stb.? Ezektől az apróságoktól most tekintsünk el a példa kedvéért.

Ez így, szavakkal leírva elég kuszának tűnhet, a nem túl szabatos megfogalmazás algoritmizálása nehézkes. Nézzük azonban az alábbi állapotgráfot, mely formalizmusával egy lépéssel közelebb juttat bennünket a megoldáshoz:



♣ Már kis problématér esetén is sokat segíthet egy beszédes ábra

A kiinduló állapot a „Billentyűre vár". Ha pont érkezik, akkor a „'.'-nál tart" állapotba kerülünk, bármely egyéb billentyű leütése esetén a kezdőállapotban maradunk. Ide térünk vissza minden olyan esetben, amikor nem a soron következő betűt, és nem is pontot gépel be a felhasználó *('.' után N, N után E, stb.)*. Ezt jelölik az ábrán látható szaggatott nyilak. Pont esetén mindig újrakezdődik a vizsgálat. Ha az utolsó helyes karakter, a 'T' is megérkezik, akkor a számláló értékét eggyel inkrementáljuk, és folytatjuk a vizsgálatot. *(Az egyszerűség kedvéért feltételezzük, hogy a leütött karakterek folyamatosan, végtelen ideig érkeznek, ezért nem jelöltük külön a végső, vizsgálatot befejező állapotot.)*

Láthatjuk, hogy már ilyen egyszerű probléma esetén is mekkora könnyebbséget jelent egy sokatmondó ábra. Nagy problémamatereknél még többet segíthet valamilyen grafikus szemléltetés használata, egyrészt saját magunk számára, másrészt munkatársaink illetve a megrendelő fél felé történő kommunikáció céljából.

Modellező eszközök kiválasztása

Az előttünk álló feladat nem más, mint hogy kiválasszuk azt a módszertant, és a hozzá tartozó eszközöket, amelyek igényeinknek leginkább megfelelnek, és a leghatékonyabban tudjuk használni. Ez esetenként nem könnyű feladat, de nem is lehetetlen…

Az nyilvánvaló, hogy valamilyen szabványos, egységesített megoldást kell keresnünk, hiszen semmit nem érünk az egész modellel, ha több idő telik el a magyarázásával és megértésével *(megértetésével)*, mint a teljes rendszer felépítésével. Az elterjedt, közismert módszerek egyrészt már kellően kiforrottak, felépítésük logikus és érthető, másrészt jó esélyünk van arra, hogy partnerünk is ismeri, így nem lesz nehéz a közös nyelvezet megtalálása.

Itt most nem célom konkrét modellezési módszertanok és eszközök bemutatása, ezekre későbbi cikkeimben részletesen kitérek. Néhány szempontot azonban kiemelnék, amelyek segíthetik a későbbi választást. Először is fontosak előismereteink, tehát ha már kellően jó barátságban vagyunk pl. az UML-lel, akkor ez jó kiindulási alap a továbblépéshez. Lényeges, hogy egyeztessünk a csoport többi tagjával, akik velünk együtt dolgoznak a projekten, és megegyezzünk egy egységes modellező eszközben, melyet mindenki elfogad és ismer *(vagy elsajátít időközben)*. Azonban ne csak a már megszerzett ismeretek legyenek a fő tényezők, hiszen az újdonságokra, új technológiákra, módszerekre mindig nyitottaknak kell lennünk *(főleg ha hosszútávon azok valamilyen pluszt, például idő- és/vagy energiamegtakarítást is jelenthetnek számunkra)*. A későbbiekben ismertetésre kerülő módszerek *(pl. UML, SSADM, Petri hálók)* szakmai szempontból lényegesen más tulajdonságokkal rendelkeznek, más-más céllal születtek. Az *UML* például rendkívül jó szemléltetésre, specifikációra és dokumentálásra, több nézetből is vizsgálhatjuk a rendszert, diagramjai áttekinthetőek, közérthetőek. Használatát rendkívül sok fejlesztőeszköz támogatja.

Petri-hálókkal konkurrens, aszinkron, elosztott, párhuzamos, nemdeterminisztikus és/vagy sztochasztikus rendszerek, workflow-k, gyártási folyamatok modellezése válik roppant egyszerűvé. *(A fenti fogalomáradattól nem kell megijedni, néhány hónap múlva már mindenki pontosan érteni fog mindent :.)* A módszer nagy hátránya azonban, hogy a problématér növekedésével a háló mérete is robbanásszerűen növekszik. Az SSADM *(Structured Systems Analysis and Design Method)* elkülönült egységekre osztja az informatikai rendszerek fejlesztéséhez tartozó lépéseket, és dinamikusan idomul a különböző jellegű feladatokhoz. Kidolgozásakor négy alapelvet helyezett középpontba a brit CCTA *(Central Computer and Telecommunications Agency, UK)*:

1. legyen önellenőrző
2. kipróbált és elterjedt módszereket alkalmazzon
3. legyen alakítható
4. könnyen el lehessen sajátítani.

Oldalakon keresztül lehetne sorolni a szempontokat az egyes metódusok mellett és ellen, de úgy gondolom, hogy a konkrét módszertanok ismerete nélkül ez felesleges időpazarlás lenne. Bevezetésként legyen elég ez a néhány figyelemfelkeltő gondolat, és ígérem, a következő hónapokban fény derül mindenre!

Molnár Ágnes
agnes.molnar@t-systems.com

Patchwork:

Scriptomatic

A Scriptomatic fejlesztői tudják a választ az élet legégetőbb kérdéseire: "why do some system administrators get fancy cars, yachts, and Rolex watches? It's because they know how to write WMI scripts, and you don't!" Ez a cikk hozzásegít álmaid jachtjához, mivel képessé tesz WMI scriptek írására.

Sokan vannak, akiknek időről-időre szükségük van egy-egy WMI scriptre. Ilyenkor ha nem tudjuk *(vagy nem akarjuk...)* önállóan megírni, elkezdünk keresgélni a TechNet Script Centerben. Hogy ezt ne kelljen mindig megtenni, a Scriptomatic fejlesztői egy olyan hipertext-alkalmazást *(Hypertext Application)* készítettek, melynek segítségével WMI *(Windows Management Instrumentation)* scripteket „szerkeszthetünk" Visual Basic nyelven.

A fejlesztők egyrészt azért választották éppen a HTA *(Hypertext Application)* formátumot, mert olyan scriptalapú alkalmazást szerettek volna létrehozni, amelyet akár egy Notepad segítségével is meg lehet írni. Másrészt egy szokványos, weboldalba ágyazott WMI script esetén folyamatosan figyelmeztető üzenettel találnánk magunkat szemben, amelyben arról értesülnénk, hogy WMI scriptek futtatása nem biztonságos. A HTA-k átcsúsznak a „not safe for scripting" figyelmeztetésen, más biztonsági beállításon nem, így távoli gépekről is csak akkor kaphatunk információt, ha azon a gépen rendszergazdai jogokkal rendelkezünk.

Ezek után nézzük, hogyan is működik a Scriptomatic. Futtatáskor először betölti az elérhető WMI osztályok neveinek kollekcióját egy legördülő listába. Innen kiválaszthatjuk azt az osztályt, amelyről információt szeretnénk lekérni.

Amint ezt megtettük, a program el is készíti a scriptet, mely segítségével az adott osztály tulajdonságainak és metódusainak értékét kaphatjuk meg. Felhasználóként nekünk csak annyi a dolgunk, hogy azokat a sorokat, melyekre nincs szükségünk, töröljük. Majd a Run gombra kattintva ellenőrizhetjük az elkészített scriptet futás közben. Ha elégedettek vagyunk az eredménnyel, akár el is menthetjük scriptünket.

Ha az elkészített scriptet nem a sajátgépen, hanem egy távoli gépen szeretnénk futtatni, akkor az

StrComputer = "."

sorban a pont helyére a távoli gép nevét kell írnunk.

A Srciptomatic azonban nem tökéleteses, több olyan igény is van melyet ez a verzió nem elégít ki.

Első ilyen hiányosság, hogy csak bizonyos osztályok kerülnek betöltésre. A fejlesztők szűrőfeltételek alapján válogatták ki azokat az osztályokat, amelyeket a gyakrabban használtak közé soroltak, így azok az osztályok, amelyek nem adnak vissza adatot (mint például a CÍM osztályok), nem kerültek a legördülő lista elemei közé.

A megjelenített osztályoknak a következő feltételeknek kell megfelelniük:

1. adjon vissza adatot;
2. a root\cimV2 névtérben helyezkedjen el;
3. a neve „Win32_"-vel kezdődjön.

Így ebben az alkalmazásban a több száz WMI osztálynak „csak" 98%-ából válogathatunk. A kimaradt osztályok száma azonban Windows.NET Server esetén jóval túlnőhet a 2%-on, mivel ebben az operációs rendszerben több hasznos osztály a root\cimV2 névtéren kívül helyezkedik el.

Aki szeretne a szűrőfeltételeken változtani, nyissa meg a scriptomatik.hta fájlt egy tesztőleges szerkesztővel *(akár Notepad is lehet)*, és írja át a megfelelő részt. Például ha névteret szeretnénk változtatni, a következő soron kell egy kicsit alakítanunk:

```
Set objWMIService = GetObject("winmgmts:\\" &
strComputer & "\root\cimv2")
```

A Scriptomatic segítségével sajnos nem tudjuk lekérdezni azokat az értékeket, amelyeket tömbökben tárol a rendszer. Ilyen például a Win32_Printer objektum Capabilities tulajdonsága, amely egy tömböt ad vissza. A fejlesztők ígérete szerint a következő verzió már kezelni fogja a tömböket is. A Scriptomatic nem értelmezi a visszaadott adatokat. Így például ha a számítógépre telepített nyomtatók állapotát kérdezzük le, azt kapjuk vissza, hogy az A nyomtató 4-es állapotban van a B nyomtató pedig 7-esben, mivel a rendszer egész számként tárolja ezeket az értékeket. Ha tudjuk, hogy ilyenkor a 4=Printing és a 7=Offline, akkor tudjuk értelmezni a kapott információt - a Scriptomatic azonban ezt nem teszi meg. Hasonló esetekben, amikor egy értéket nem tudunk értelmezni, a WMI SDK-ban megtaláljuk a megfelelő információt. A Scriptomatic csak értéket ad vissza, metódus futtatására nem alkalmas. Ha például egy service-ről szeretnénk megtudni valamit, arra kiválóan alkalmas, de ugyanezt a service-t elindítani vagy leállítani ebből az eszközből már nem tudjuk.

A Scriptomatic csak a WMI-vel működik együtt. A WMI segítségével információt kaphatunk többek között számítógépünkről, a számítógépen futó szoftverekről, a perifériákról. Amit a WMI segítségével nem tehetünk meg *(és így a Scriptomaickal sem)*, nem kérhetünk információkat az Active Directoryban található adatokról. Így ezzel az eszközzel nem kérhetjük le a felhasználó, a szervezeti egységek vagy éppen a biztonsági csoportok adatait.

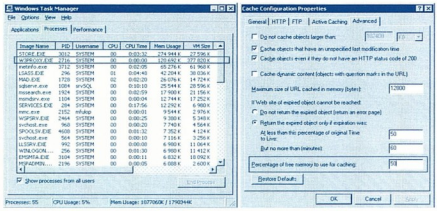
Bármire ugyan nem használható ez a kis eszköz, azonban segítség lehet azoknak, akik gyorsan szeretnének rövidebb scripteket írni, vagy éppen most mélyednek el a WMI scriptek írásának rejtelmeiben.

Végül nézzük, milyen rendszerkövetelményei vannak a Scriptomatic futtatásának. Windows XP, Windows 2000 és Windows .NET Server operációs rendszerek esetén nincs egyéb feltétele az alkalmazás használatának. Windows NT 4.0 esetén azonban Windows NT 4.0 SP6-re, legalább Internet Explorer 5.0-re, WMI-re és a WSH 5.6-ra van szükség. Windows 98 operációs rendszer esetén szintén Internet Explorer, WMI és WSH 5.6 a feltétel.

Borsi Katalin
bobo@netacademia.net

Dupla KV

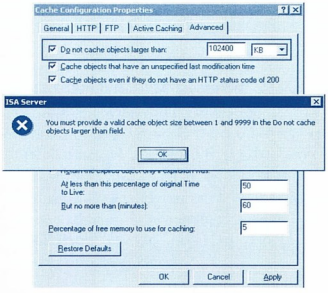
K: *Tisztelt Szerk.! Három kérdésem lenne a Microsoft Internet Security and Acceleration (köznapi nevén ISA) Serverrel kapcsolatban. Itt van mindjárt az első: kiszolgálónkon az Exchange és a webproxy áldáz harcot vív a drága memóriáért. Egyszerűen nem értem, miért kell fél giga memória egy nyomorult gyorsítótárnak, miközben a felhasználóink hetente nem töltenek le annyit. Az Exchange meg közben „éhezik"…*



♣ ***Csendélet a kiszolgálón: harc a memóriáért***

V: Ez bizony így van, ha harc, legyen harc. Az ISA Server gyorsítótára bizony alapértelmezésben ráveti magát a rendelkezésre álló fizikai memória felének megfelelő méretű területre, és legfeljebb abban enged, hogy annak egy részét hagyja kilapozni a lemezre. Ha úgy gondoljuk, hogy tudnánk a memóriát nemesebb célra használni, fogjuk viszsza a proxy falánkságát. Indítsuk el az ISA felügyeleti konzolját, majd a konzolfában kattintsunk jobb gombbal a Cache Configuration sorra. Válasszuk a Properties parancsot, majd a megjelenő dialógusablak utolsó *(Advanced)* oldalának alján módosítsuk tetszés szerint a „Percentage of free memory to use for caching:" értékét. A változtatás akkor lép életbe ha újraindítottuk a web proxy szolgáltatást.

K: *Nagyszerű. A következő kérdés a gyorsítótár tartalmával kapcsolatos. Szeretném meghatározni, hogy a proxy legfeljebb mekkora objektumokat tároljon a cache-be. Kitaláltuk, hogy 100 megabájtnál nagyobb méretű dolgokat nem tárolnánk. Meg is találtam a beállításra szolgáló mezőt (a fent is említett Cache Configuration Properties párbeszédablakban), de amikor érvényesíteném a változtatásokat, csúnya hibaüzenet fogad…*



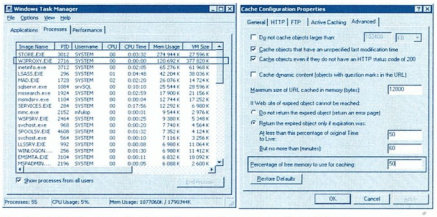
♣ ***A beállíthatatlan beállítás***

V: Ez bizony hiba, és legjobb tudomásunk szerint a mai napig fennáll. A probléma oka - állítólag - az, hogy az adatok tárolására szolgáló mező maximális értéke 9999 lehet, csak a fejlesztők megfeledkeztek arról az apróságról, hogy a szorzóértéket külön megpróbálják eltárolni. Úgy vesszük, hogy az értéket mindig kilobájtban kell értelmezni. A konzol is ezt csinálja: ha 9999 kB-nál nagyobb értéket adunk meg, a mező értékét felszorozza, *(lásd: 100x1024 = 102400)*, aztán pedig nem tud vele mit kezdeni…

K: *Ejha! És nem lehet ezzel valamit kezdeni? A Proxy 2.0-ban legalább bele tudtunk nézni a cache lemezterületébe, és ha kellett, kézzel törölni azt, ami nem tetszett. Nincs valamilyen módszer arra, hogy lássuk, mi került be a gyorsítótárba, ne adj' Isten, esetleg törölhessük is a nem kívánt objektumokat?*

V: Szerencsére van ilyen eszköz. Egyrészt, ha nagyon nagy a gond, a teljes gyorsítótár-könyvtár tartalmát is törölhetjük *(például az ISA Server telepítő CD-jén található \Support\Tools\Admin_Tools\Deleteall.vbs script segítségével)*, de ugyanitt találunk egy kis eszközt is, amivel böngészhetünk is a cache-ben.

Az ISA CD \Support\Tools\Troubleshooting könyvtárában található cachedir.exe programocskát másoljuk oda, ahol az ISA Server programfájljai vannak. Ha készen vagyunk, indítás után a következőhöz hasonló kép fogad minket:



♣ ***CacheDir.exe: kalandozások a gyorsítótárban***

A programocska induláskor gyors fastruktúrát épít a tárban található objektumokból, így webcím alapján keresgélhetünk a cache-ben. Az ablak alján jelzi azt is, hogy hány objektumot tartalmaz a gyorsítótár, és ezek összesen mekkora helyet foglalnak el. Látható a tárolt objektumok neve, mérete, tartalomtípusa, lejárati ideje, sőt, bele is kukkanthatunk a fájlok tartalmába *(binárisan)*. Ha pedig egy objektumot szeretnék kipucolni a gyorsítótárból, kattintsunk a nevére jobb gombbal, majd válasszuk a „Mark as Obsolete" parancsot. A listát egyébként szöveges fájlba is menthetjük, ehhez adjuk ki a következő parancsot:

```
cachedir.exe -dump filen0v.txt
```

K: *Hogyan tudom megállapítani egyszerűen, hogy kinek van joga betárcsázni a cégnél üzemelő RAS kiszolgára?*

V: A Windows NT és Windows 2000 Resource Kit része a rasusers.exe segédprogram, amelynek segítségével könnyedén meg lehet állapítani kinek van joga a behíváshoz. A rasusers <tartomány név> parancs a tartományban levő összes felhasználó nevét kilistázza, akinek joga van a betárcsázáshoz.
A rasusers \\<kiszolgáló> parancs pedig megmutatja az adott kiszolgálóra ki tárcsázhat be.
Ha már itt tartunk, szeretném felhívni a figyelmet a raslist.exe-re, amellyel megtalálhatjuk a hálózatban levő összes RAS kiszolgálót.

(Forrás: www.ntfaq.com)

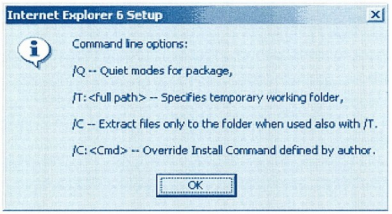
K: *Hogyan lehet Windows XP alól Exchange 2000-et kezelni? Az Exchange System Management Tools-t nem tudom telepíteni, mert azt mondja, hogy a Windows 2000 Administration Tools nincs telepítve. Persze, mert a Windows .NET-es adminpak.msi van fent.*
V: Egy lehetséges megoldás, hogy egy Windows 2000 kiszolgálóra *(akár az Exchange 2000 kiszolgáló is lehet ez)* Terminal Servicest telepítünk. Így Windows XP alól terminál kapcsolaton keresztül lehet az Exchange 2000-et adminisztrálni.
Egy másik megoldás, amikor Windows XP-re telepítjük az Exchange System Managert. Ennek lépései a következők:

1. Le kell szedni a Windows .NET Administration Pack-ot, ha fent van a Windows XP-n.
2. Telepíteni kell a Windows 2000 Administration Pack-ot
3. Ezután lehet telepíteni az Exchange 2000 System Managert
4. Utolsó lépésként pedig fel kell telepíteni a Windows .NET Administration Pack-ot.

Nem egészen szabályos, de működő megoldás.

(Forrás: Windows 2000 levelezési lista)

K: *Szeretném letölteni a Internet Explorer 6 SP1-es verzióját, de csak az ie6setup.exe található, ami letöltené és rögtön telepítené is a kiválasztott komponenseket. Nekem viszont csak le kellene tölteni az egészet. Milyen megoldás létezik?*
V: Érdemes megnézni az ie6setup.exe paramétereit:

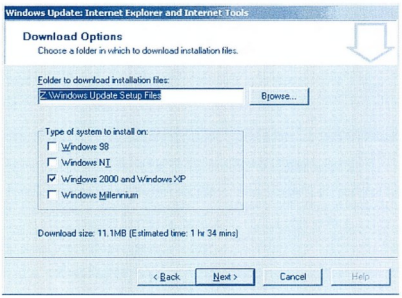


♣ ***ie6setup.exe /? parancs eredménye***

Ez ugyan nem árul el mindent, de a lényeg az, hogy a /c kapcsolót kell használni ahhoz, hogy a telepítő varázslót más paraméterekkel tudjuk futtatni. A következő parancsot kell kiadni ahhoz, hogy a telepítőkészlet letöltődjön:

```
ie6setup.exe /c:"ie6wzd.exe /d /s:"
```

Ennek hatására elindul a telepítővarázsló, majd megjelenik az alább is látható ablak, ahol választhatunk mely operációs rendszerekhez szeretnénk letölteni az Internet Explorert.



♣ ***Internet Explorer telepítővarázsló***

Más megoldásként a
http://corporate.windowsupdate.microsoft.com siteról is letölthető egészben az Internet Explorer.

(Forrás: Windows 2000 levelezési lista)

K: *Hogyan lehet rávenni az Outlook 2002-őt arra, hogy amikor minimalizálom az ablakot akkor a „System Tray" területre kerüljön az ikon?*

V: A regisztrációs adatbázis módosításával van lehetőség a kérdés megoldására.
A következőt kell a registrybe felvenni:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\10.0\
Outlook\Preferences

Value: MinToTray
Data: 1
Type: REG_DWORD
```

Azaz, ha a megadott elérési úton a MinToTray kulcs értékét 1-re állítjuk az Outlook a „System Tray" területre kerül minimalizálás után. A registry módosítása után csupán az Outlookot kell újraindítani.

(Forrás: www.ntfaq.com)

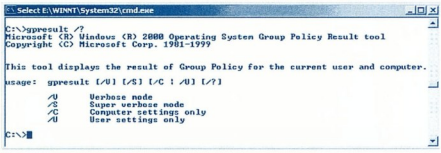
K: *Outlook 2002-őt használunk vele együtt a MSN Messenger is fut a gépen. Minden alkalommal amikor be akarom zárni a Messengert azt az üzenetet kapom, hogy egy program használja a Messengert, ezért nem engedi bezárni. Van lehetőség arra, hogy ezt mégis megtegyem?*
V: Alapértelmezésben az Outlook 2002 és az MSN Messenger működése integrált, de szétválasztható.
Ehhez a következőt kell tenni:

1. Az Outlookban el kell navigálni a Tools menü Options beállításokhoz
2. Itt az Other feliratú tulajdonságlapon az „Enable Instant Messaging in Microsoft Outlook" jelölőnégyzetből ki kell venni a pipát.
3. A beállítások érvényre jutásához újra kell indítani az Outlookot.

(Forrás: www.ntfaq.com)

K: *Elég sok problémám van abból, hogy a tartományban beállított csoportos házirendek nem, vagy nem teljesen hozzák a várt eredményt. Hogyan tudom megállapítani egy munkaállomáson, hogy egy adott felhasználóra és a gépére éppen milyen beállítások érvényesek?*

V: Egy megoldás a Windows 2000 Resource Kit-ben található gpresult.exe. Megfelelően paraméterezve részletes információhoz juthatunk az éppen érvényes beállításokról. A paramétereket a szokásos /? Kapcsolóval kaphatjuk meg:



A „/s" kapcsolóval kaphatjuk a legrészletesebb információt az éppen érvényes beállításokról.
Ha csak a felhasználóhoz tartozó házirendek végeredményét szeretnénk megkapni akkor /c kapcsolót kell használni.
Érdemes a parancs kimenetét egy fájlba átirányítani, majd a fájlt elemezni.

K: *Szeretném végérvényesen megszüntetni a fájltitkosítás lehetőségét Windows XP munkaállomásokon. Azt szeretném, hogy senki ne tudjon titkosítani fájlokat a munkaállomásokon. Nem találtam erre vonatkozó beállítást a grafikus felületen. Van egyáltalán lehetőség letiltani a titkosítást?*

V: Windows 2000 és Windows XP esetén is a registry módosításával lehet kiiktatni a titkosítást. A következőt kell módosítani a registryben:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\
Windows NT\CurrentVersion\EFS

Value: EfsConfiguration
Data: 1
Type: REG_DWORD
```

Vagyis az EfsConfiguration kulcs 1-re állításával lehet letiltani a titkosítást. A beállítás érvényre jutásához újra kell indítani a számítógépet.
Ezután egy fájl titkosításakor a következő hibaüzenet jelenik meg:

♣ ***Hibaüzenet miután letiltottuk a fájltitkosítást***

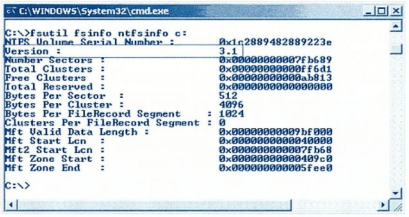
(Forrás: www.ntfaq.com)

K: *Hogy lehet megállapítani, hogy egy NTFS milyen verziójú? Window NT-n, Windows 2000-en és Windows XP-n is más a verziója az NTFS-nek. De honnan tudom, hogy egy partíción melyik NTFS van?*

V: A Windows 2000 és a Windows XP része az fsutil segédprogram, ami többek közt elárulja az NTFS verziót is.
A következő parancsot kell kiadni parancssorból.

```
fsutil fsinfo ntfsinfo <k tet>:
```

Az alábbi képen látszik a parancs eredménye:



♣ ***Információk egy NTFS partícióról***

A képen a pirossal is bekeretezett sor mutatja az NTFS verziószámát, jelen esetben egy Windows XP-ről van szó, az NTFS verziószáma 3.1. Windows 2000 NTFS verziószáma 3.0, a Windows NT 4.0 esetén pedig 1.2.

(Forrás: www.ntfaq.com)

K: *Windows 2000 operációs rendszeren használjuk a beépített Quota szabályozást. Most egy másik partícióra kell költöztetni az adatokat, de nem szeretném a Quota információkat kézzel átvinni. Át lehet vinni az összes Quota beállítást az egyik partícióról a másikra valahogy?*

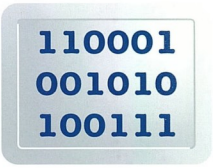
V: Igen. Az egyik partíción exportálni kell a qouta beállításokat egy fájlba, a másik partíción pedig lehet importálni. A pontos menete a következő:

1. A köteten - amelyről exportálni szeretnénk a beállításokat - context menüből kiválasztjuk a tulajdonságokat *(jobb klikk - Properies)*
2. A Quota tulajonságlapon a Quota Entries gombra kell kattintani
3. Itt a Quota menüből az Export parancsot kell választani
4. Meg kell adni a fájl nevét, amelybe a Quota információk kerülnek
5. A másik köteten - ahova át szeretnénk tenni a beállításokat, az előzőekhez hasonlóan el kell navigálni a Quota bejegyzésekhez.
6. Itt a Quota menüben Importot kell választani, majd meg lehet adni az előbb exportált fájl helyét és nevét.

Az import folyamat során az operációs rendszer figyelmeztet, ha a célköteten már léteznek olyan quota bejegyzés, amely ütközik a fájlban levő információkkal. Ilyenkor választhatunk melyik beállítást szeretnénk megtartani.

(Forrás: www.ntfaq.com)

Tanulj fiam, mert megbux!



Máris kezd lecsukódni a szemem, ha az egyetemi évek alatt végighallgatott előadásokra gondolok. Körmölés másfél órán keresztül, nehogy ne tudjam azt az egy-két kulcsmondatot, ami a jegyzetben nem fellelhető, a sikeres vizsgának azonban elengedhetetlen feltétele. Sokakat foglalkoztat a kérdés, hogyan lehetne úgy tanítani, hogy az ne kényszer legyen a diákok számára, hanem olyan kaland, amire szívesen emlékeznek, és alig várják a következő órát. Ez olyan, több tudományterületet átfogó kérdés, amelyre a válasz más volt tíz évvel ezelőtt, és valószínűleg más lesz tíz év múlva is. Annyi mindenesetre bizonyos, hogy a műszaki fejlődés a választ jelentős mértékben befolyásolja, egyrészt az elsajátításra váró ismeretek rendíthetetlen növekedésével, másrészt a segédeszközök végtelen tárházával.

A Microsoft és az MIT 1999 októberében úgy döntött, hogy kísérletet tesznek megfelelni e filozófiai magasságokat érintő kérdést. Az első „ötéves terv" keretében létrehozták az iCampus nevű projektet az egyetemi oktatás hatékonyságának információ-technológiai eszközökkel való javítására. Az MIT tagjai, a diákokat is beleértve kutatási javaslatokat tehetnek, amiket egy bizottság bírál el. Eddig csak a diákok kezdeményezései alapján közel félmillió dollárt osztottak szét. Íme egy kis ízelítő a legsikeresebb próbálkozásokból.

TEAL

E négy betű a Technology Enhanced Active Learning rövidítése, és egészen jól fedi a valóságot. Képzeljünk el egy nagy termet, legyen mondjuk 15 x 20 méteres. A falakon nyolc nagy méretű kivetítő látható. A terem közepén álló asztalt az előadó bitorolja, akit körbevesz további 13 kerek, hozzávetőleg két méter átmérőjű asztal, egyenként 9 diákkal, 3 számítógéppel és egy elektronikus táblával, amelyre például kivetíthető az asztalon folyó munkát megörökítő videokamera képe. A számítógépek közötti kapcsolat magától értetődő. A szereplők sorát két instruktor zárja, akik a gyakorlatok során az előadóval együtt segítik az elakadt diákokat, vagy ha éppen valami érdekeset látnak, a többieknek is megmutatják.

Számoljunk csak: 13 x 9 = 117 diák ülhet egyszerre a teremben. Emlékeim szerint ez nem több, mint egy átlagos egyetemi évfolyam. Ugyanakkor, ha jól emlékszem az első és utolsó előadásokat leszámítva, alig 20-30 hallgató tisztelte meg jelenlétével az előadókat. Ezzel szemben John Belcher, a TEAL terem egyik tervezőjének óráin a beiratkozott hallgatók 85 százaléka vesz részt. A siker oka nyilvánvaló. Az előadó nem untatja másfél órán keresztül elmélettel a jelenlévőket, tulajdonképpen csak felvezeti a problémamegoldó és laborgyakorlatokat, amik az előadások savát-borsát képezik. A terem ennyiben általános, de a jelenlegi eszközrendszerrel kifejezetten az elektromágnesesség szemléletes oktatása volt a cél, az asztalokon álló kísérleti berendezések erőtereinek és kölcsönhatásainak számítógépes modellezésével.

iLab

Emlékeimben fel-felsejlik a gimnáziumi kémiaszertár sötét zúga. Sosem tudtuk meg, hogy démonok, vagy kémcsövek tárházául szolgált. Értesüléseim szerint sokan voltak ezzel így, mert az iskolák költségvetése nem tette lehetővé, hogy minden diákból méregkeverő váljék. Talán ez is hozzájárult nem túl fényes vegyértékeimhez, miközben a természettudományok iránt kifejezetten nagy érdeklődést tanúsítottam.

Az iLab nagyjából ezt a problémát hivatott megoldani. Az ilyen labor igazi labor, nem csak egy számítógépes modell. A kulcsot a mágikus i betű jelenti: a labor az Interneten keresztül távvezérelhető, méréseket lehet végrehajtani, adott keretek között új kísérleteket lehet tervezni, az eredmények lekérdezhetők. Az iLab non-stop üzemel, tehát a hagyományos laborokkal szemben a kihasználtsága jelentősen, gyakorlatilag 100 százalékosra növelhető. Persze sokkal többe is kerülnek, de nem tartom kizártnak, hogy egy teljes életciklus-elemzésben az iLab szakítaná át elsőnek a célszalagot.

Del Alamo professzor 5 éve indította útjára iLabjét, kezdteben mikroelektronikai eszközök mérésére, de a koncepció később sikeresen alkalmazhatónak bizonyult a gépészet, a vegyészet és a statika területén is. A professzor statisztikái szerint az iLabban elvégzendő házi feladatokat a diákok 95 százaléka oldja meg, a hagyományos feladatoknál tapasztalható 80 százalékkal szemben, és ezen túl még az osztályzatok is jobbak.

Úgy gondolom a megoldás ebben az esetben is kézenfekvő: a problémamegoldás testközelbe került, és a diákok nagyfokú önállóságot kaptak.

SVAS

Az iCampus a humán területre is betört. Az SVAS *(Shakespeare Video Annotation System)* Shakespeare drámák gyűjteménye, és még több. Egyszerre több videó is nézhető, így könnyebben feltárhatók a rendezők interpretációi, megoldásai közötti hasonlóságok vagy különbségek, és ezekről azonnal elektronikus jegyzet is készülhet. Egy ilyen rendszer, ahogy az iLab is, rugalmasabbá teszi az időbeosztást, és a világ távoli pontján élő diákokból is verbuválható virtuális osztály.

There is no spoon!

A teljesség igénye nélkül hadd soroljam fel mégegyszer az előnyöket: hatékonyabb *(szemléletesebb, felszabadultabb)* oktatás és tanulás, rugalmas időbeosztás, a földrajzi korlátok ledöntése, az oktatási területek dinamikusabb átalakíthatósága a kor igényei szerint.

Ez mind szép és jó, ha nem feledkezünk meg az érme másik oldaláról sem. A technika legalább olyan mértékben köt bennünket gúzsba, mint amilyen lehetőségeket tár fel előttünk. Ugye, senki sem számít arra, hogy új felfedezéseket tehet egy olyan laborban, amelyben a szabadságfokainak számát a vezérelhető elemek száma szabja meg! Egyszóval, végre előttünk lehet a kémcső, de véletlenül sem törhetjük össze.

Zacco@fw.hu

A cikkben szereplő URL:

[1]: http://research.microsoft.com/features/icampus_overview.asp

2150 - Biztonságos Windows 2000 hálózat tervezése

A 2150-es tanfolyam hiányt pótol a Microsoft tanfolyamok között. A Windowsba az évek során – szerencsére – lassan beszivárogtak az alapvető biztonsági funkciók, de a Windows biztonságossá tételéhez azért még sok helyen akad tennivaló. A tanfolyam során nemcsak a Windows 2000 új biztonsági elemeit ismerjük meg, hanem bemutatjuk a rendszer gyengeségeit is. Olyan információkat is igyekszünk átadni, amelyek a hivatalos tankönyvekben nem szerepelnek, de az Interneten nyílt titokként bárki hozzájuk férhet. Amíg nem vagyunk tisztában az általunk védeni kívánt rendszer gyengeségeivel, megvédeni sem tudjuk azt.

Szükséges előismeretek
Windows 2000 üzemeltetési ismeretek.

Alapozó tanfolyamok
OSI, 1560

Továbbképző tanfolyamok
2150, 2159

Összefoglalás
A továbbiakban e tanfolyam összefoglalóját, valamint néhány vizsgakérdést olvashatnak. Részletesebb, napokra lebontott tematika, valamint a kérdések megoldása a http://nate.netacademia.net/temat/nate2150.asp címen olvasható.

Fülöp Miklós
MCSE, MCT, MCSA

Tanfolyami összefoglaló:

1. fejezet: Bevezető, a veszélyek felismerése
2. fejezet: A Windows 2000 biztonsági elemeinek áttekintése
3. fejezet: Rendszergazdai hozzáférés tervezése
4. fejezet: Felhasználói fiókok és jogosultságok tervezése
5. fejezet: A Windows 2000 védelme
6. fejezet: A fájl- és nyomtatókiszolgálók védelme
7. fejezet: A kommunikációs csatornák védelme
8. fejezet: A nem-MS ügyfelek hozzáférése
9. fejezet: Távoli felhasználók biztonságos hozzáférése
10. fejezet: Távoli telephelyek biztonságos hozzáférése
11. fejezet: Interneten keresztül érkező felhasználók biztonságos hozzáférése
12. fejezet: Biztonságos Internet-hozzáférés a belső felhasználók részére
13. fejezet: A hálózat kiterjesztése a partnervállalatok felé
14. fejezet: Nyílt kulcsú infrastruktúra tervezése
15. fejezet: Biztonsági terv készítése

Kapcsolódó vizsgák
70-220 Designing Security for a Microsoft Windows 2000 Network

Tesztkérdések - szükségem van-e erre a tanfolyamra?

A tanfolyamhoz tartozó vizsga során esettanulmányok alapján kell megadnunk a választ a kérdésekre. A következőkben bemutatunk egy rövid esettanulmányt, majd néhány hozzá kapcsolódó kérdést.

Esettanulmány: X cég munkatársak kiközvetítésével foglalkozik. A cég központja *(150 alkalmazottal)* Budapesten van, fiókirodái szerte az országban, elszórva találhatók. A fiókirodák mindegyike 128 kbit sávszélességű bérelt vonallal kapcsolódik a központhoz, saját kiszolgálókkal nem rendelkezik. Közvetlen Internet-kapcsolat csak a központban található. Az alkalmazottak az ország bármely pontjáról elérhetik postafiókjukat, a központban működő Exchange Server 5.5 Outlook Web Access szolgáltatásának köszönhetően. A központi számítógéphez – ha szükséges – Terminal Services segítségével férnek hozzá. Budapesten ezen kívül még intranetes web-, valamint RAS-kiszolgáló és UNIX-on futó Oracle adatbáziskiszolgáló is üzemel. A cégnél minden kiszolgálót Windows 2000-re frissítenek *(a központ és minden telephely közös Active Directory tartományba tartozik)*, beállítanak egy újabb RAS- *(ezúttal RRAS-)* kiszolgálót, valamint minden ügyfelet digitális PKI tanúsítványokkal látnak el – ez megkönnyíti a biztonságos felhasználóazonosítást.

1. Milyen biztonsági megoldás(oka)t kell bevezetnünk a központi telephelyen?

   A. EFS

   B. Digitális tanúsítványkezelés *(PKI)*

   C. Titkosított adatforgalom

   D. PAP felhasználóazonosítás

   E. Kétirányú azonosítás

2. Az átállás után milyen biztonsági módszerrel férhetnek hozzá a vidéki munkatársak a intraneten található adatokhoz?

   A. NTLM

   B. SSL és basic felhasználóazonosítás

   C. Kerberos

   D. MS-CHAP

3. Milyen biztonsági beállításokat választanánk a fiókirodák és a központi kiszolgáló közötti hálózati forgalom titkosításához?

   A. Kerberos azonosítás, 3DES titkosítás, AH protokoll

   B. Kerberos azonosítás, 3DES titkosítás, ESP protokoll

   C. Tanúsítványalapú azonosítás, 3DES titkosítás, AH protokoll

   D. Tanúsítványalapú azonosítás, 3DES titkosítás, ESP protokoll

   E. Jelszavas azonosítás, 3DES titkosítás, AH protokoll

   F. Jelszavas azonosítás, 3DES titkosítás, ESP protokoll

2159

2150

1560

OSI

A tanfolyam időpontjai	A tanfolyam ára
2002. november 18-tól január 13-ig minden héten hétfő délután 14-18-ig	135.000,- Ft

2349 - A .NET keretrendszer programozása C# nyelven

Aki már akár saját maga, akár a 2124-es tanfolyam segítségével túljutott a C# programozási nyelv alapjain, megértette azokat az objektumorientált és komponensalapú fejlesztési elveket, amelyek a C# magját alkotják, annak jöhet a nagyobb falat, a .NET keretrendszer *(.NET Framework)*, illetve ezen belül a .NET Class Library *(osztálykönyvtár)* megismerése.
Ez a tananyag a keretrendszerről szól. A programozók idejük java részében a keretrendszer szolgáltatásait veszik igénybe egy-egy funkció megvalósításakor. Kihasználásához és eléréséhez jelentős segítséget és fejlesztési sebességet ad a Visual Studio.NET, de az a keretrendszer ismerete nélkül csak kövérre hízott kódvarázslóként fogyasztja a merevlemezünket.
Mint a tematikából látható, nagyon nagy anyagrészekkel ismertet meg ez a tanfolyam. Ezeket elsajátítva már biztos kézzel el lehet kezdeni a fejlesztési és tervezési munkákat. A .NET keretrendszer akkora falat, amit lehetetlen teljes egészében feldolgozni *(3500 publikus osztály, félmillió metódus)* akár tanfolyami keretek között, akár önállóan. Ez a tanfolyam viszont alkalmas olyan alap átadására, amelyen elindulva a hallgatók már képesek a további részletek és újabb témakörök önálló elsajátítására. Mondjuk úgy, hogy megtanít .NET-ül gondolkodni, szemléletet ad a további fejlődéshez. Az MCP-vizsgák mindegyike kérdez olyan információkat, amelyekre ez a tanfolyam képes választ adni, így vizsgafelkészítőnek is jól használható anyagról van szó.

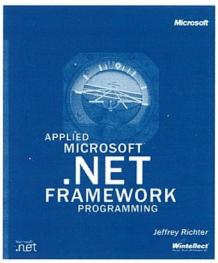
Szükséges előismeretek
A C# gördülékeny használata, de minimum a C# kódok gyors értelmezése alapvetően fontos az anyag megértéséhez. Előzményként javasolhatjuk a 2124 - Programozás C# nyelven tanfolyamunkat.

Programozási nyelv
C#

Alapozó tanfolyamok
2124

Továbbképző tanfolyamok
2350, 2389, 2524, 2555

Ajándékkönyv
Applied Microsoft .NET Framework Programming



Összefoglalás
A továbbiakban a tanfolyam összefoglalóját, valamint néhány vizsgakérdést olvashatnak. Részletesebb, napokra lebontott tematika, valamint a kérdések megoldása a http://www.netacademia.net/training/course.aspx?id=2349 címen olvasható.

Soczó Zsolt
MCSE, MCDBA, MCSD, MCT

Tanfolyami összefoglaló:

1. fejezet: A .NET keretrendszer áttekintése
2. fejezet: A menedzselt futtatási környezet felépítése
3. fejezet: .NET komponensek fejlesztése
4. fejezet: Telepítés és verziózás (assembly elmélet és gyakorlat)
5. fejezet: Common Type System
6. fejezet: Típusok használata
7. fejezet: Stringek, tömbök, kollekciók
8. fejezet: Delegate-ek és események
9. fejezet: Memória- és erőforráskezelés
10. fejezet: Adatfolyamok és fájlok kezelése
11. fejezet: Hálózati programozás, internetelérés
12. fejezet: Serialization: a remoting alapja
13. fejezet: Remoting és XML webszolgáltatások
14. fejezet: Opcionális, a hallgatókkal egyeztetve: Többszálú programok és aszinkron programozás

15. fejezet: Opcionális, a hallgatókkal egyeztetve: Menedzselt és natív kód együttműködés

16. fejezet: Opcionális, a hallgatókkal egyeztetve: Adatelérés ADO.NET-tel

17. fejezet: Opcionális, a hallgatókkal egyeztetve: Attribútumok

Kapcsolódó vizsgák

70-315 Developing and Implementing Web Applications with Microsoft Visual C#™ .NET and Microsoft Visual Studio .NET

70-316 Developing and Implementing Windows-based Applications with Microsoft Visual C#™ .NET and Microsoft Visual Studio .NET

70-320 Developing XML Web Services and Server Components with Microsoft Visual C#™ and Microsoft and the Microsoft .NET Framework

Tesztkérdések - szükségem van-e erre a tanfolyamra?

1. Egy más gyártó által fejlesztett titkosító komponenst szeretnénk C# programból használni. A titkosító metódus egy Win32-es DLL-ben van megvalósítva, és egy titkosítandó szöveget vár paraméterül, második paraméterében pedig visszaadja a titkosított szöveget. Hogyan kell definiálni a PInvoke számára a paramétereket?

A. Encrypt(String nyilt, String titok)

B. Encrypt(String nyilt, StringBuilder titok)

C. Encrypt(StringBuilder nyilt, String titok)

D. Encrypt(StringBuilder nyilt, StringBuilder titok)

E. Encrypt(String nyilt, ref String titok)

F. Encrypt(String nyilt, out String titok)

2. Ha egy osztályban felüldefiniáljuk a System.Object Equals metódust, akkor az == is ennek megfelelően fog működni?

A. Igen, mert a háttérben a fordító visszavezeti az == hívást az Equals-ra

B. Nem, külön felül kell definiálni az ==-t is

C. Az == egy beépített operátor, nem lehet felüldefiniálni

3. Melyik kollekció indexelhető string típussal?

A. ArrayList

B. Hashtable

C. SortedList

D. StringDictionary

Megoldások magyarázattal: http://www.netacademia.net/training/course.aspx?id=2349&s=quiz

MesterQurzus Kiskonferenciák

Minden hónap utolsó csütörtökén kerül megrendezésre MesterQrzusunk, ahol elsőkézből, a szerzőtől szerezhet mélyebb ismereteket a tech.net-ben megjelenő cikkek nehezebb témaköreiről, az előadások után pedig tantermünkben lehetőséget nyújtunk a hallottak gyakorlati kipróbálására is! Önnek csak az a dolga, hogy idejében regisztrálja magát a weben, a következő oldalon: http://technet.netacademia.net/mesterq

2002. október 31. 14:00 óra — 15.000,- Ft

Group Policy ***(Dorner Csilla)***

Amiről kevesen beszélnek:

- Csoportos házirendek vegyes környezetben
- Miért más a csoportos házirend terminal services kiszolgálón?
- Hogyan lehet saját gyártású házirendet készíteni?

Objektumorientált alapelvek +DP ***(Soczó Zsolt)***

- A .NET Framework objektumorientált környezet, így használatához és .NET-es alkalmazások írásához elengedhetetlenül szükséges az OOP alapelvek ismerete. A fontosabb letisztázandó fogalmak: osztály, objektum, identitás, egyenlőség, egységbezárás, metódus, mező, jellemző, öröklődés, polimorfizmus, aggregálás, kompozíció, delegálás, virtuális tagok.

Golyóálló webkiszolgálók ***(Fülöp Miklós)***

- A Windows 2000/XP/.NET Server webkiszolgálójának védelme
- Új biztonsági tudnivalók, elemek, eszközkészlet

A .NET Web Server újdonságai

Gyakorlat: HTTPS, IISLockD, URLScan, IPSec csomagszűrés

2002. november 28. 14:00 óra — 15.000,- Ft

DNS és DDNS ***(Fülöp Miklós)***

- A Windows 2000 DNS-kiszolgáló és ügyfelek működése
- Dinamikus DNS-regisztráció Windows környezetben
- Az Active Directory és a DNS

C# (Soczó Zsolt)

- A C# a .NET talán legfontosabb nyelve. A nyelv milyen elemei támogatják a .NET Framework programozását? Mire készüljön fel egy C++ programozó? Mit kell megtanulni egy VB-programozónak a C# használatához? Érdemes-e áttérni rá? Ezeket a kérdéseket gombolyítjuk fel ebben az előadásban.

Remote Installation Services: áldás vagy csapás? ***(Dorner Csilla)***

- Mi minden kell a működéséhez?
- Mit lehet, és mit nem lehet RIS-sel telepíteni?
- RIS és a Windows XP

Gyakorlat: RIS-kiszolgáló felkészítése munkaállomások telepítésére

2003. január 30. 14:00 óra — 15.000,- Ft

Terminal Services ***(Dorner Csilla)***

- Hol használható igazán?
- Méretezés: Mekkora vas kell alá? Hány felhasználót bír el?
- Furfangos TS-licencelés
- A Windows .NET TS újdonságai

XML, XSL ***(Soczó Zsolt)***

- Az XML már rengeteg technológia legalapvetőbb összetevője. Miért terjedt így el a szakmában, mire találták ki, mire használják? Milyen járulékos technológiákat szült? Hogyan és mire használható az XSLT?

LDAP, LDIF ***(Fóti Marcell)***

- Az LDIF-szabvány
- Tömeges címtármódosítás export/import segítségével
- Sémabővítés
- ADSI-gyorstalpaló

Gyakorlat: címtármódosítás LDIF-fájllal, ADSI-scripttel

Tesztek megoldásai:

81. oldal:	83. oldal:
1: B, C	1: B
2: B	2: B
3: D	3: B, C, D

Ugróiskola- a NetAcademia egyedi, hiánypótló tanfolyamai

A 2002. tavaszától útjára indított Ugróiskola előadássorozat célja, az azonnal hasznosítható ismeretanyag átadása. Aki beugrik hozzánk, az úgy jut tovább élete következő mezőjére, hogy valami hasznos szerszám-eljárás birtokába jutott.

2002. őszétől új képzéseket indítunk az Ugróiskola keretében, melyek:

OSI Hálózati ismeretek a szabványok tükrében
SQL Halmazorientált világ: az SQL nyelv
OOP Az objektumorientált fejlesztés alapelvei
SMTP Elektronikus levelezés az Interneten
WWW Webes technológiák
SECU Betörésvédelem
NM Hálózatmonitorozás
DIS Katasztófaelhárítás
RIS Felügyelet nélküli telepítés
XP Windows XP nagyvállalati környezetben
AD Active Directory mélyvíz
SCR Scripting

http://www.netacademia.net/ugrosuli/

Ismerje meg a jövő fejlesztőeszközét és technológiáit!

Microsoft .NET fejlesztői képzéssorozat a SZÁMALK Továbbképzésnél

Kedvezményes konstrukció • Rugalmas időbeosztás • A kezdő szinttől a haladó ismeretekig

Párhuzamosan sajátíthatja el a Visual Basic .NET és a Visual C# nyelv használatát

Bizonyára Ön is hallott a Microsoft forradalmian új, .NET névre keresztelt technológiájáról, melynek szerves részét képezi a fejlesztői oldalról a **Visual Studio .NET** termék. A SZÁMALK Továbbképzés által összeállított tanfolyamsorozaton a hallgatók párhuzamosan ismerhetik meg a két új programnyelv és fejlesztői környezet, – a Visual Basic .NET és a Visual C# .NET – felépítését, nyelvi szintaktikáját, elemeit, újdonságait, a Windows alapú és webalapú alkalmazások és szolgáltatások fejlesztését.

A Microsoft .NET fejlesztői képzés hat darab hivatalos Microsoft tanfolyamra épül, összesen öt hétben. A tanfolyamok minden hónap egy-egy hetében kerülnek megrendezésre, vagyis 3–5 napot jelentenek havonta.

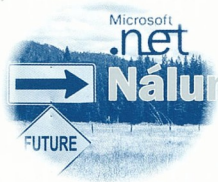
Microsoft CERTIFIED Solution Developer

- Programozás Microsoft Visual Basic .NET / C# és Visual Studio .NET környezetben
- Adatkezelés (SQL 2000 lekérdezések, ADO.NET)
- Windows alapú alkalmazások fejlesztése Visual Basic .NET és C# környezetben
- Webalapú alkalmazások fejlesztése Visual Basic .NET és C# környezetben
- XML és webalapú szolgáltatások tervezése Visual Basic .NET és C# környezetben

Microsoft CERTIFIED Application Developer

A képzések az okleveles *alkalmazásfejlesztői (MCAD)* és *fejlesztőmérnöki (MCSD)* cím megszerzéséhez szükséges vizsgákhoz is segítséget nyújtanak.

Kezdési időpont: október 14-e.



Kezdési időpont: Nálunk biztosan indul!

Minőség, egyéneknek kedvező konstrukciók, cégeknek partneri kedvezmények!

Microsoft CERTIFIED Technical Education Center

SZÁMALK TOVÁBBKÉPZÉS

SZÁMALK

☎ 203-0304/3050 (Simon Ferenc) **www.szamalk.hu/tisza** 1115 Budapest, Etele út 68.